Extending the Nanos6 Runtime with a Lightweight Profiling Infrastructure

Autor
Antoni Navarro Muñoz

Director
Eduard Ayguadé Parra (DAC)

Co-director
Vicenç Beltran Querol (BSC-CNS)

Grau en Enginyeria Informàtica

Especialitat
Enginyeria de Computadors

19 de Gener de 2017

Agraïments

Crec convenient agrair primerament a Eduard Ayguadé i Vicenç Beltran, els meus directors de projecte, per donar-me l'oportunitat de formar part de l'equip d'un projecte a gran escala com és Nanos6 i deixar-me realitzar el meu treball final de grau en les instal·lacions del BSC i no només això, sinó també guiar-me quan ha sigut necessari i proporcionar-me els medis per tal de fer possible aquest projecte.

També vull agrair l'ajuda proporcionada per Josep M. Pérez, que en tot moment m'ha resolt qualsevol dubte sobre el runtime i m'ha guiat per a desenvolupar el projecte amb una bona praxi.

Voldria donar les gràcies també a tot el personal del BSC que m'ha ajudat al llarg del projecte, sigui resolent dubtes sobre el compilador o fent-me sentir part de l'equip.

Per últim també agrair a totes les noves amistats creades en el centre, a la meva família i a la meva parella el suport constant.

Resum

Després d'un breu resum sobre el model de programació OmpSs i els elements que formen el seu entorn, com és el runtime Nanos6 i el compilador Mercurium, aquest document mostra el desenvolupament portat a terme per dotar Nanos6 amb un mòdul de mesura per fer profiling de tasques, un mòdul de predicció i un mòdul anomenat *autofinal* que farà ús dels mòduls previs per incrementar el rendiment del runtime, així com el disseny i la implementació de tots aquests mòduls. Es començarà dissenyant el mòdul de mesura, definint tots els conceptes i característiques que és capaç d'obtenir d'execucions. Més endavant es modificaran el model i el runtime per tal de suportar noves clàusules com la clàusula *cost*. Després s'implementarà el mòdul de prediccions que s'alimentarà de les dades proveïdes pel mòdul de mesura i, per últim, el mòdul autofinal que farà ús explícit del mòdul de predicció però a la vegada farà ús implícit de tots els nous mòduls o clàusules dissenyats. Un cop finalitzada la part de disseny i implementació, s'avaluarà el rendiment del runtime extés amb diferents *benchmarks*, prèviament analitzant el temps afegit a causa de les modificacions fetes al runtime.

Abstract

After a brief about the OmpSs programming model and all the elements which create its environmennt, such as the Nanos6 runtime or the Mercurium compiler, this document will show the development of several modules in order to extend Nanos6 with a measurement module, a prediction module and an *autofinal* module which will use all of the previously mentioned modules. First the measurement module will be designed, while defining all the concepts and characteristics that it can extract from executions. After that, the model and runtime will be modified in order to support new clauses such as the *cost* clause. The implementation of the prediction module which accepts data provided by the measurement module will follow. Once the aforementioned modules are designed and implemented, the autofinal module will be implemented and it will use all the previous modules either implicitly or explicitly. Afterwards, a performance evaluation of the extended runtime will be done using different *benchmarks*, previously analyising the overhead added due to modifications to the runtime.

Glossari

benchmark Aplicacions utilitzades per mesurar el rendiment de sistemes o components de sistemes. ii, 9

directives En programació, una directiva o pragma és una construcció de llenguatge que especifica a un compilador, assemblador o intèrpret com processar l'entrada rebuda. vi, 1, 3, 4

profiling El profiling és una forma dinàmica d'anàlisi que pot mesurar, per exemple, temps o recursos en aplicacions. 3

runtime En programació, una llibreria runtime és un conjunt de rutines de baix nivell usades per els compiladors amb la finalitat d'invocar comportaments d'un entorn runtime, inserint crides a la llibreria runtime dins l'executable compilat. ii, 1, 2, 4–9, 11–18

scheduling En la computació, l'scheduling o les polítiques d'scheduling són els mètodes pels quals el treball d'execució és assignat als recursos disponibles. 6, 55, 56

thread Un thread o fil d'execució és la unitat més petita de programació que pot ser utilitzada pels schedulers d'un sistema operatiu. 4

Índex

1 Context **1**
1.1 Introducció . . . 1
1.2 Actors Implicats . . . 1

2 Estat de l'art **3**
2.1 OmpSs . . . 3
2.1.1 Definició . . . 3
2.1.2 Funcionament i model d'execució . . . 4
2.1.3 Entorn . . . 5
2.2 Projectes relacionats . . . 6
2.2.1 Paraver . . . 6
2.2.2 Dimemas . . . 6
2.2.3 Extrae . . . 6
2.2.4 Nanos6 . . . 6
2.3 Conclusions - Estat de l'Art . . . 7

3 Abast i Objectius del projecte **8**
3.1 Motivació . . . 8
3.2 Requeriments . . . 8
3.3 Objectius . . . 8
3.4 Abast . . . 9
3.5 Riscs i possibles solucions . . . 10
3.5.1 No disponibilitat de *MareNostrum* . . . 10
3.5.2 Problemes amb el maquinari . . . 10
3.5.3 Augment de temps no previst per a realitzar certes tasques . . . 10
3.5.4 Errades en la implementació . . . 10
3.5.5 Errades en projectes relacionats . . . 11

4 Metodologia **12**
4.1 Mètodes de treball . . . 12
4.2 Eines de seguiment . . . 12
4.3 Mètodes de validació . . . 13
4.4 Avaluació del resultat final . . . 13

5 Planificació temporal **14**
5.1 Descripció de les tasques . . . 14
5.1.1 Gestió del Projecte . . . 14
5.1.2 Estudi del model de programació i el runtime . . . 14
5.1.3 Anàlisi general . . . 15
5.1.4 Creació de l'entorn . . . 15
5.1.5 Disseny i implementació . . . 15
5.1.6 Avaluació del rendiment . . . 16
5.1.7 Memòria final . . . 16
5.2 Dependències entre tasques . . . 16
5.3 Previsió temporal . . . 17
5.4 Recursos . . . 17
5.4.1 Recursos Hardware . . . 17
5.4.2 Recursos Software . . . 18
5.4.3 Recursos Humans . . . 18

5.5 Valoració d'alternatives i pla d'acció 18

6 Gestió Econòmica **20**
6.1 Pressupost dels Recursos 20
6.2 Costos Indirectes 21
6.3 Imprevistos i Contingències 21
6.4 Pressupost Final 22
6.5 Control de Gestió 22

7 Sostenibilitat i Compromís Social **24**
7.1 Matriu de Sostenibilitat 24
7.2 Dimensió Econòmica 24
7.3 Dimensió Social 25
7.4 Dimensió Ambiental 25

8 Nanos6 **26**
8.1 Cicle de vida del Runtime 26
8.2 Clàusules Suportades 27

9 Disseny i Implementació **29**
9.1 Mòduls de Monitoratge: Mesura i Predicció 29
9.1.1 Instrumentació 29
9.1.2 Obtenció de les Mètriques 31
9.1.3 Estructures del Runtime 32
9.1.4 Estructures de Timing 33
9.1.5 Implementació dels Mòduls de Mesura i Predicció 34
9.2 Profiling de Tasques 35
9.3 Clàusula Cost 36
9.4 Mòdul Autofinal 38
9.4.1 Clàusula Final 38
9.4.2 Autofinal 39
9.4.3 Modificacions a conseqüència del Mòdul Autofinal 41
9.4.4 Implementació del Mòdul Autofinal 42

10 Resultats **43**
10.1 Overhead Afegit pel Mòdul de Mesura 43
10.2 Rendiment del Runtime Extés 45
10.3 Casos d'ús de Nanos6 51

11 Conclusions **52**

12 Treball Futur **53**
12.1 Separació de tasques per *Buckets* en Cost 53
12.2 Scheduler 54
12.3 Problemàtica amb Weak Dependencies 54

13 Revisió del projecte **55**

14 Referències **57**

Apèndixs **59**

Índex de figures

1 *Successió de Fibonacci en C++ amb directives OmpSs* 4
2 *Esquemes dels models d'execució Thread-Pool i Fork-Join* 5
3 *Entorn del model OmpSs* 5
4 *Dibuix del cicle de vida del runtime.* 26
5 *Codis d'exemple amb dependències.* 28
6 *Esquema conceptual de la relació entre els mòduls implementats.* 29
7 *Relació entre tasques i labels en el mòdul de mesura.* 32
8 *API del mòdul de mesura.* 34
9 *Benchmarks exemple amb clàusula cost.* 36
10 *Solver de l'Algoritme NQueens usant Clàusula Final.* 38
11 *Codi actual de la decisió sobre si una tasca és final o no.* 40
12 *Comprovacions al crear una tasca a causa del mòdul autofinal.* 41
13 *Overhead afegit mesurat amb diferents benchmarks* 43
14 *Fibonacci(30) executat a MN3* 45
15 *Fibonacci(30) executat a Mont-Blanc* 46
16 *Algoritme de Fibonacci amb clàusula final explícita.* 46
17 *MergeSort(100.000.000) executat a MN3* 47
18 *MergeSort(100.000.000) executat a Mont-Blanc* 48
19 *Algoritme de MergeSort amb clàusula final explícita.* 48
20 *NQueens(15) executat a MN3* 49
21 *NQueens(15) executat a Mont-Blanc* 50
22 *Solver de l'algoritme d'NQueens amb clàusula final explícita.* 50
23 *Mètode de l'algoritme d'NQueens que mira si la posició escollida és amenaçada.* . . 51
24 *Captura de pantalla de l'execució dels jocs de prova de Nanos6.* 51
25 *Gràfic d'execucions de mergesort amb el cost unitari associat* 53
26 *Taula de Gantt amb les dates d'inici i finalització, la durada, els nivell de riscs i els recursos de les tasques del projecte.* 59
27 *Diagrama de Gantt amb les tasques del projecte, les dependències entre elles i fites de temps relacionades amb el projecte.* 60
28 *Diagrama del pas d'estats en tasques per tal d'obtenir mètriques de timing.* 61

Índex de taules

1 *Dependències entre les tasques del projecte.* 16
2 *Previsió temporal de les tasques del projecte.* 17
3 *Costos dels recursos humans.* 20
4 *Costos dels recursos software.* 20
5 *Costos dels recursos hardware.* 21
6 *Estimació de costos indirectes.* 21
7 *Pressupost total del projecte.* 23
8 *Matriu de sostenibilitat del TFG.* 24
9 *Diferents valors de la variable d'entorn NANOS6 i la configuració establerta* . . . 30
10 *Exemple d'ús del mòdul de mesura amb la clàusula cost* 37

1 Context

Des de fa més de 20 anys, el progrés en velocitat de xarxes, arquitectures de multiprocessadors i sistemes distribuïts mostren que el futur de la computació és el paral·lelisme.

Amb l'arribada de l'era de la computació paral·lela, l'atenció dels camps de la informàtica relacionats amb el paral·lelisme s'ha anat focalitzant en dissenyar models de programació que s'adaptin a la computació d'altes prestacions i a sistemes de gran escala com els supercomputadors.

Aquests models de programació comporten grans beneficis com l'estalvi de temps, que deriva en estalvi de recursos humans, la possibilitat de resoldre problemes més complexos i l'ús de hardware paral·lel, entre d'altres.

En les següents subseccions s'introdueix aquest projecte i s'especifica el personal involucrat en el desenvolupament d'aquest.

1.1 Introducció

Aquest projecte és un Treball Final de Grau del grau d'Enginyeria Informàtica, més concretament de l'especialitat d'Enginyeria de Computadors de la Facultat d'Informàtica de Barcelona (Universitat Politècnica de Catalunya). Ha sigut desenvolupat en col·laboració amb el Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS).

OmpSs és un model de programació paral·lela desenvolupat pel Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS). Aquest model és una extensió del model de programació paral·lela OpenMP[1] amb les directives i funcionalitats més destacades del model de programació StarSs[2], també desenvolupat pel BSC. Dues d'aquestes funcionalitats més destacades són el suport al paral·lelisme asíncron i l'heterogeneïtat en dispositius com acceleradors.

L'entorn d'OmpSs s'integra pel compilador Mercurium[3], el runtime Nanos++ i eines d'anàlisi. Més endavant s'entra en detall en tots els components que formen l'entorn del model d'OmpSs.

La finalitat d'aquest projecte és extendre el runtime Nanos6[4] amb un mòdul de mesura lleuger i un mòdul que pugui fer prediccions de temps d'execució, canviar el comportament de les execucions de tasques per obtenir-ne més rendiment i que s'adapti al suport, per part del runtime, del model de programació paral·lela d'OmpSs[5].

1.2 Actors Implicats

El desenvolupament d'aquest projecte té una sèrie de persones implicades directament o indirectament i en diferents nivells d'implicació, des del més implicat: el desenvolupador, fins als que en trauran profit: els beneficiaris. A continuació s'exposen els actors implicats.

- **Desenvolupador:** El projecte només constarà d'un desenvolupador, jo mateix. Tot i que el runtime té més persones implicades, jo sóc la persona activa pel que fa al projecte, ja que sóc l'encarregat de tot el que comporta la creació dels mòduls necessaris, és a dir, el codi, els tests o proves d'aquest, la documentació i la recerca d'informació.

- **Director i codirector:** Tant el director com el codirector juguen un paper molt important en el projecte, ja que s'encarreguen de guiar i supervisar al desenvolupador per assegurar que compleix amb tots els objectius marcats dintre del temps establert, convocant reunions de seguiment sempre que sigui convenient.

- **Personal de suport:** Atès que el projecte del desenvolupador s'integra en altres projectes en desenvolupament, com per exemple el del runtime Nanos6, el personal que treballa en aquests projectes serveix de suport al desenvolupador, ja sigui resolent dubtes sobre el funcionament del runtime o aportant coneixements i suggerint millores.

- **Barcelona Supercomputing Center:** El centre en si s'implica de dues maneres en el projecte. Per una banda, aporta al desenvolupador tots els recursos necessaris per a la realització del projecte, tant *software* com *hardware*, ja sigui el portàtil amb el qual treballarà o l'ordinador de taula. Per un altra banda, com s'explica tot seguit, també juga el paper de beneficiari.

- **Beneficiaris:** El beneficiari directe serà tot l'equip del Barcelona Supercomputing Center que estigui en projectes relacionats amb Nanos6, ja que el runtime tindrà una sèrie de mòduls que permetran saber informació important sobre les execucions i fins i tot millorar-les amb prediccions, pel que estalviarà temps. Indirectament, també es veuran beneficiats els usuaris del runtime, ja que podran fer servir aquests mòduls perquè les seves aplicacions tardin menys en executar-se.

2 Estat de l'art

Aquest projecte pretén implementar una sèrie de mòduls que, en part, permetin mesurar el temps d'execució d'aplicacions, cosa que moltes eines de *profiling* ja permeten, com per exemple les creades pel BSC que s'explicaran en aquesta secció més endavant. Tanmateix, aquestes eines no permeten accedir a les mesures en temps real, cosa de vital importància per un dels objectius d'aquest projecte: les prediccions per a poder millorar l'execució d'aplicacions.

Seguidament s'expliquen tots els elements que integren el context del projecte en el seu estudi, siguin conceptes o eines.

2.1 OmpSs

2.1.1 Definició

OmpSs és un model de programació compost per un conjunt de directives i mètodes que poden ser usats juntament amb un llenguatge de programació d'alt nivell amb la finalitat de crear aplicacions concurrents. Aquest model de programació integra característiques de la família del model de programació StarSs, el qual va ser creat pel grup *Programming Models* del departament de Computer Sciences del Barcelona Supercomputing Center (BSC).

L'objectiu d'OmpSs és proveir d'un entorn productiu per desenvolupar aplicacions per a sistemes de computació d'altes prestacions (*HPC*). Hi ha dos conceptes bàsics que fan que OmpSs sigui aquest entorn productiu: rendiment i facilitat d'ús. Pel que fa a rendiment, els programes desenvolupats amb OmpSs haurien de ser capaços de mostrar un rendiment competitiu comparat amb altres models de programació que tenen com a objectiu les mateixes arquitectures.

Un dels projectes més ambiciosos és extendre el model de programació OpenMP amb noves directives, clàusules i serveis per a poder donar suport al paral·lelisme amb flux de dades asíncron i l'heterogeneïtat de dispositius com *GPUs*. Per heterogeneïtat de dispositius s'entén els sistemes que usen més d'un tipus de processador. Aquests sistemes guanyen en rendiment no només per integrar un sol tipus de processador sinó processadors amb coprocessadors amb capacitats de processament especialitzades per a cert tipus de tasques.

Un dels conceptes d'OmpSs no quantificable és la senzillesa d'ús, anteriorment mencionada. En aquest aspecte OmpSs s'assembla a OpenMP, ja que integra el mateix patró d'ús: permet paral·lelitzar programes amb el simple fet d'introduir directives o clàusules. Tanmateix, l'ús d'OmpSs no implica haver de paral·lelitzar zones de codi explícitament, cosa que si que és necessari en OpenMP. OmpSs se centra més en quines parts de codi vol paral·lelitzar l'usuari. La figura 1 mostra un petit codi en C++ on es pot apreciar la senzillesa d'ús dels dos models de programació.

```
void fibonacci(int index, int *resultPointer) {
  if (index <= 1) {
    *resultPointer = index;
    return;
  }

  int result1, result2;

  #pragma oss task shared(result1)
  fibonacci(index-1, &result1);

  #pragma oss task shared(result2)
  fibonacci(index-2, &result2);

  #pragma oss taskwait
  *resultPointer = result1 + result2;
}
```

Figura 1: *Successió de Fibonacci en C++ amb directives OmpSs*

2.1.2 Funcionament i model d'execució

OmpSs es basa en tasques i dependències. Una tasca es defineix com la unitat bàsica de treball que representa una instància específica de codi executable. Les dependències fan possible que l'usuari especifiqui el flux de dades d'un programa, per tal que el runtime pugui fer servir aquesta informació per determinar si l'execució paral·lela de dues tasques tindria conflictes en l'ús de dades.

El sistema del runtime, en començar l'execució del programa d'usuari, crea un conjunt de fils d'execució o *threads*. Aquest conjunt de *threads* es compon per un *thread master* i d'altres addicionals. El thread master executa el codi d'usuari seqüencialment com si hi haguera una tasca implícita envoltant tot el programa, mentre que tots els altres threads esperen fins que hi ha tasques concurrents disponibles per ser executades.

Quan qualsevol thread es troba una clàusula de tasca, una nova tasca explícita és generada. Les execucions d'aquestes tasques explícites són assignades a un dels threads del conjunt. No obstant això, l'execució de la tasca pot ser immediata o retardada, en el cas que hi hagin dependències no resoltes o el thread no estigui disponible.

Aquest model d'execució s'anomena *thread-pool*, important no confondre'l amb el model *fork-join* usat per OpenMP. El funcionament bàsic del model *fork-join* consisteix en la divisió de l'execució del programa d'usuari en punts designats per tal d'assolir paral·lelisme i l'agrupació d'aquestes divisions en punts subseqüents. Aquestes seccions paral·leles poden ser dividides recursivament fins que una certa granularitat de tasques s'ha assolit. La figura 2 il·lustra la diferència entre els dos models mencionats.

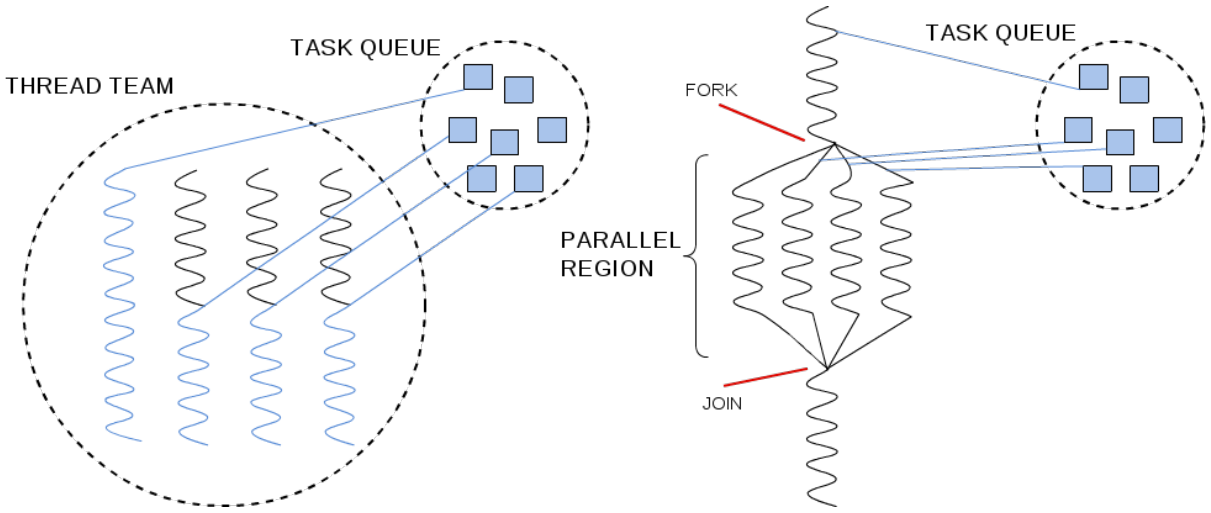

Figura 2: *Esquemes dels models d'execució Thread-Pool i Fork-Join*

2.1.3 Entorn

Com s'ha mencionat en anteriors seccions, OmpSs té un entorn integrat per diferents elements, entre ells el compilador Mercurium, el runtime Nanos++, del qual es treballarà amb la versió 6 (Nanos6) en aquest projecte i diferents eines d'anàlisi. Entre aquestes eines cal destacar *Paraver*[6], *Dimemas*[7] i *Extrae*[8].

Mercurium és una infraestructura de compilació que suporta els llenguatges *C, C++ i Fortran*. Bàsicament es fa servir en l'entorn del runtime Nanos per implementar OpenMP però també es fa servir en més models.

Atès que aquest projecte s'enfoca a l'anàlisi i té com a base de treball el runtime de Nanos6, s'aprofundeix més en Nanos6 i les eines d'anàlisi de l'entorn de OmpSs en les següents seccions, per tal d'extreure conclusions sobre el motiu d'aquest projecte.

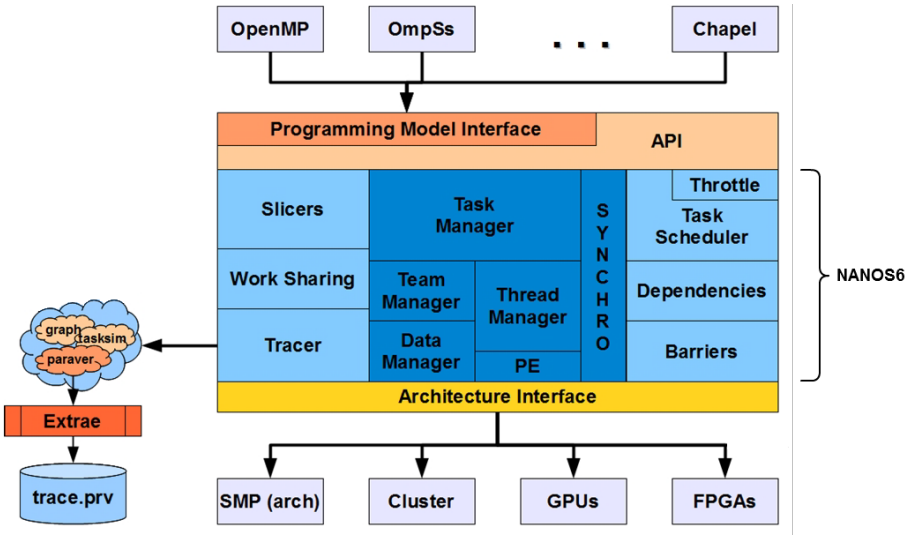

Figura 3: *Entorn del model OmpSs*

2.2 Projectes relacionats

En seccions anteriors han sorgit noms d'eines d'anàlisi creades pel Barcelona Supercomputing Center. Seguidament s'entrarà en detall en les seves característiques i d'altres projectes relacionats.

2.2.1 Paraver

Paraver va ser creat per respondre a la necessitat de tenir una percepció global del comportament de les aplicacions mitjançant inspeccions visuals. Aquesta eina permet veure traces d'execució d'aplicacions amb diferents tipus de vistes o configuracions amb el simple fet de refrescar la vista actual. La granularitat de l'anàlisi arriba fins a nanosegons, cosa que assegura una visió completa sobre el que passa en l'execució en tot moment.

2.2.2 Dimemas

Dimemas és una eina d'anàlisi de rendiment per programes de pas de missatges. Permet a l'usuari crear i optimitzar aplicacions paral·leles mentre va proveint prediccions del rendiment en la màquina on s'està executant. *Dimemas* genera traces que són compatibles amb *Paraver*, fent possible que l'usuari pugui examinar amb profunditat el rendiment.

2.2.3 Extrae

Extrae és un *package* dedicat a generar traces *Paraver* per a una anàlisi de l'aplicació un cop ja s'ha executat. És una eina que usa diferents mecanismes per inserir sondes en l'aplicació amb motiu de recollir informació sobre rendiment.

2.2.4 Nanos6

El projecte de Nanos6 és el que es farà servir com a base d'aquest projecte. Nanos6 és una versió del runtime Nanos++ així que en aquesta secció es referirà al runtime com a Nanos. Nanos és una llibreria designada per a servir com a suport en entorns paral·lels. Es fa servir sobretot per donar suport al model OmpSs però també té mòduls per donar suport a OpenMP i Chapel[9].

Aquest runtime ofereix diferents serveis per a suportar el paral·lelisme de tasques usant sincronitzacions basades en dependències de dades. També ofereix suport per mantenir la coherència en diferents espais d'adreces com per exemple amb GPUs o nodes de clúster.

El propòsit principal de Nanos és ser usat en recerca d'entorns de programació parallela. És per això que ha sigut dissenyat per ser extensible a través de *plugins*: les polítiques *d'scheduling*, implementacions de barreres, la capa d'instrumentació i el nivell d'arquitectura només per mencionar-ne uns quants.

2.3 Conclusions - Estat de l'Art

Tal com s'ha exposat en les seccions 2.1 i 2.2.4, per la realització d'aquest projecte es faran servir el model de programació OmpSs i el runtime Nanos6. En relació a OmpSs, s'hauran de fer algunes modificacions per tal de suportar clàusules introduïdes per aquest projecte, com és la clàusula cost explicada més endavant. Per últim, el runtime es modificarà necessàriament per afegir el mòdul de mesura i el de predicció, tanmateix, no haurien d'obstruir gaire en el disseny del runtime.

Pel que fa a altres runtimes i projectes semblants, mai s'havia afegit un mòdul de mesura que alimenti altres mòduls, com el de predicció, en temps d'execució. Només s'havien implementat a forma d'instrumentació per a tenir dades sobre l'execució d'aplicacions al final d'aquestes. És per això que la frontera del coneixement en aquest àmbit de projecte és visible en exemples com les eines d'anàlisi descrites en les seccions 2.2.1, 2.2.2 i 2.2.3, així que, el fet de crear mòduls de mesura no és res que no s'hagi vist abans, però el fet d'usar aquestes dades per predir comportaments en temps d'execució i millorar execucions sí.

3 Abast i Objectius del projecte

En aquesta secció es parlarà de tot el que inclou l'abast del projecte, és a dir, el perquè del projecte o motivació, els objectius principals, la solució proposada, els mètodes de treball que s'aplicaran així com eines usades per assolir la metodologia escollida, riscos que poden aparèixer i solucions a aquests.

3.1 Motivació

Com s'ha mencionat abans, l'entorn d'aquest projecte i altres projectes relacionats tenen ja les seves pròpies eines d'anàlisi de rendiment. No obstant això, aquestes eines tenen una utilitat *post-mortem*, és a dir, fins que l'execució no s'ha finalitzat el rendiment no pot ser examinat. És per això que es va consensuar a través de tot l'equip de suport al desenvolupador i el desenvolupador mateix, crear un mòdul de mesura que permetés accedir a la informació escaient en temps d'execució, per tal de poder prendre decisions sobre la manera més escaient d'executar certes tasques i fer prediccions de rendiment.

3.2 Requeriments

Aquest projecte té com a objectiu principal dotar el runtime de Nanos++, més concretament la versió Nanos6, de mòduls per mesurar, predir i prendre decisions en temps d'execució, és a dir, conèixer en tot moment com s'està comportant l'execució d'una aplicació abans que aquesta acabi. És per això que té una forta relació amb el projecte Nanos6.

A continuació es llisten tots els requeriments per assolir els objectius especificats més endavant:

- Els mètodes d'obtenció de rendiment (*timing*) han de ser lleugers per a no intervenir de manera perjudicial en l'execució d'aplicacions.
- Els mòduls creats han de ser totalment independents del runtime per no afegir dependències que puguin destruir la portabilitat del runtime.
- Per cap raó s'hauria d'afegir complexitat al runtime o qualsevol element que l'integri. Els mòduls hauran de ser tancats.
- L'estil de programació i les funcionalitats s'han d'adaptar als requisits imposats pel Barcelona Supercomputing Center i l'equip de Nanos6.
- En tot moment s'ha de portar un seguiment i un *feedback* actualitzat amb la resta de l'equip dels projectes relacionats.

3.3 Objectius

Per tal d'assolir tots els requisits i l'objectiu principal especificats en la secció 3.2 s'han de seguir els següents objectius:

- Dissenyar estructures de dades i mètodes lleugers en l'execució del runtime.

- Tenir accés en tot moment a aquestes estructures de dades en temps d'execució.
- Dissenyar tots els mòduls de manera estàtica i independent del runtime a mode d'instrumentació. D'aquesta manera serà independent de la versió del runtime i podrà activar-se només quan es desitgi.
- Dissenyar i implementar un mòdul que permeti usar informació per a millorar execucions d'aplicacions.
- Minimitzar en la mesura del possible l'ús de blocs condicionals i operacions costoses no necessàries.
- Minimitzar l'ús de regions crítiques al màxim per tal de no afegir *overhead* als fils d'execució.

3.4 Abast

Abans de començar a implementar, el desenvolupador haurà de passar per un període d'aprenentatge amb tots els elements que formen l'entorn del projecte, detallats a continuació:

- **OmpSs:** El desenvolupador haurà de familiaritzar-se amb l'entorn del model i de com s'usen les seves funcionalitats.
- **Nanos6:** A més del model OmpSs, el desenvolupador també haurà de familiaritzar-se i estudiar tot el codi del runtime, ja que és on s'inseriran els mòduls. Haurà d'observar com és l'estil de programació, adaptar-se a ell i entendre tot el codi del runtime.
- **Instrumentació:** Ja que existeix un mòdul d'instrumentació en el runtime, el desenvolupador haurà d'estudiar-lo i decidir si pot aprofitar alguna funcionalitat existent o el mòdul en la totalitat.

Al finalitzar aquesta fase d'estudi, el desenvolupador haurà de començar la recerca del millor mètode per analitzar el rendiment del runtime, és a dir, el millor mètode de capturar el temps de la manera més lleugera possible. Un cop trobat aquest mètode, haurà de dissenyar les estructures de dades que calguin per completar els mòduls esmentats anteriorment.

En tot moment el desenvolupador haurà de verificar la validesa del runtime, per tal de comprovar que les modificacions no han modificat el comportament d'execució. Per fer-ho, haurà d'executar tots els *benchmarks* proporcionats per l'equip de suport que proven la totalitat dels casos d'ús de totes les funcionalitats del runtime.

Un cop acabat el mòdul de mesura i comprovada la seva validesa i correctesa, caldrà consensuar quines són les dades a recollir que seran d'utilitat per altres mòduls i saber on localitzar aquestes dades, és a dir, a on inserir les estructures de dades.

Tan bon punt el mòdul de mesura estigui creat, localitzat i verificat farà falta crear un mòdul de predicció que usi les dades proporcionades per aquest per a millorar el rendiment del runtime. Aquesta part és la que comportarà més temps, ja que serà un bucle d'aprenentatge en què primer el desenvolupador implementarà mòduls que es nodreixin de les prediccions del mòdul de predicció, i llavors haurà d'anar refinant aquests mòduls mentre executa proves i analitza el comportament de les aplicacions i com són de bones les prediccions.

3.5 Riscs i possibles solucions

En aquesta secció s'entra en detall en tots els riscos que podrien sorgir mentre es du a terme aquest projecte, així com possibles solucions a aquests.

3.5.1 No disponibilitat de *MareNostrum*

És possible que a cause de problemes tècnics o manteniment, el supercomputador no estigui operatiu durant un període de temps, el qual pot anar des d'hores fins a dies.

Solució: En cas que sorgeixi aquest problema no seria crític, ja que només es farà servir el supercomputador per executar jocs de proves, cosa que es pot intercanviar per qualsevol altra tasca a fer.

3.5.2 Problemes amb el maquinari

Hi ha una probabilitat que el maquinari amb el qual el desenvolupador treballi s'espatlli o s'extraviï per qualsevol raó, cosa que comportaria pèrdua de temps i diners per part del desenvolupador, ja que és ell qui s'ha d'encarregar de tenir cura del material.

Solució: Aquest problema té solució, ja que es treballa amb versions de control, pel que tot el projecte té còpies al dia. Només comportaria una pèrdua de temps d'un o dos dies per culpa d'haver-ho d'instal·lar tot de nou.

3.5.3 Augment de temps no previst per a realitzar certes tasques

Algunes funcionalitats a implementar tenen complexitat, com per exemple els algoritmes de predicció i els algoritmes que faran ús de les prediccions per tal de modificar la manera en què s'executen les tasques. Això pot comportar delegar un temps extra inesperat a aquestes tasques.

Solució: L'única manera de solucionar aquest possible problema és preveure aquests augments de temps per tasques complexes i deixar marges més amplis per poder delegar més temps.

3.5.4 Errades en la implementació

A cause de la complexitat del projecte és probable que sorgeixin errors de disseny en algunes parts, cosa que pot comportar una reestructuració del codi i, conseqüentment, una pèrdua de temps molt important.

Solució: Aquest possible risc es pot evitar i detectar molt fàcilment si es porta a terme un seguiment molt detallat i acurat, cosa que s'explicarà en les seccions de mètodes de treball.

3.5.5 Errades en projectes relacionats

És poc probable que aquest risc es dugui a terme, tanmateix, cap la possibilitat de trobar errors en el runtime Nanos6 que obstrueixin aquest projecte.

Solució: En aquest cas s'haurà d'alertar a l'equip corresponent i trobar una solució com més aviat millor, ajudant si s'escau.

4 Metodologia

En aquesta secció es donen a conèixer els mètodes de treball seguits, les eines de seguiment fetes servir, els mètodes de validació i l'avaluació del resultat final.

4.1 Mètodes de treball

Amb raó de voler fer un projecte ambiciós, el temps és un recurs que no es pot malbaratar, cosa que implica seguir una metodologia que tingui com a principi bàsic la producció màxima amb el mínim temps. En aquests aspectes, s'han extret les propietats més significatives d'algunes de les metodologies més famoses entre programadors:

- ***Work-pulling:*** Aquest concepte, extret de la metodologia *Kanban*[10], té com a objectiu bàsic no ofegar als desenvolupadors amb massa feina. Bàsicament, els treballadors (o treballador en aquest cas) demanen feina quan ja han acabat l'anterior. En el context d'aquest projecte es podria veure com el desenvolupador planejant més tasques amb el director i el co-director un cop ha acabat les tasques que havia de fer amb anterioritat.

- ***Learning Orientation:*** Extret de la metodologia *Disciplined Agile Delivery* (DAD[11]). Aquesta idea té com a objectiu l'aprenentatge mentre es treballa. Té una gran relació amb el desenvolupament d'aquest projecte, ja que el desenvolupador ha d'anar aprenent de l'equip de Nanos6 per a tenir les idees clares abans d'enfrontar-se al runtime.

- ***Extreme Programming For One:*** Aquesta metodologia és una variant de la metodologia *Extreme Programming*[12] que recull les idees més importants de la versió pensada per equips. Aquesta metodologia promou la no-perfecció en el producte final, és a dir, en comptes de lliurar tots els objectius perfectament definits i a la perfecció qualsevol dia en un futur llunyà, aquest concepte pretén lliurar el que és essencial i necessari quan sigui necessari, sense malbaratar temps.

- **SCRUM[13]:** D'aquesta metodologia s'extreu un principi bàsic, el de desenvolupament incremental. Té com a objectiu treballar en el projecte per etapes amb motiu de tenir un software que es pugui provar al final de cada etapa.

4.2 Eines de seguiment

Per tal de tenir control del runtime s'usarà el gestor de control de versions *Git*[14]. Git és una eina totalment portable i amb un immens catàleg d'opcions. El grup de Programming Models del Barcelona Supercomputing Center té el seu propi *Gitlab*[15] del que es farà ús.

També es farà servir setmanalment una eina que permet enregistrar tots els avenços del projecte Nanos6, anomenada *Microsoft OneNote*[16]. Aquesta eina permet al director i al co-director tenir constància del treball fet i el que queda per fer de tots els projectes relacionats amb Nanos6, inclòs aquest.

4.3 Mètodes de validació

A cause de la complexitat del codi del projecte, és de vital importància tenir una metodologia de validació molt exigent, ja que qualsevol error no localitzat a temps en una fase anterior del projecte podria comprometre la integritat de les fases següents de manera greu. És per això que, a part d'executar tots els jocs de prova del runtime Nanos6, també es crearan tests específics per assegurar la correctesa de les funcionalitats afegides al runtime gràcies al meu projecte.

Aquests tests seran fets en *C++* i se centraran a explotar les funcionalitats del runtime extés, provant casos extrems.

A manera de validació també hi haurà reunions setmanals de l'equip de Nanos6, on el desenvolupador en formarà part i haurà de fer un resum setmanal sobre el què va implementar la setmana anterior. Així, el director i/o el co-director podran ser capaços de detectar errors i guiar al desenvolupador en el transcurs del projecte.

4.4 Avaluació del resultat final

Tan bon punt tot el codi dels mòduls a implementar estigui acabat i s'hagin posat a prova les seves funcionalitats, es procedirà a avaluar el rendiment del runtime Nanos6 amb les noves funcionalitats. Es calcularà l'impacte de l'extensió i sobretot s'analitzarà i s'avaluarà el rendiment final per veure l'impacte en el temps d'execució que tenen les prediccions.

5 Planificació temporal

En aquesta secció s'entra en detall en la descripció i la previsió temporal de les tasques a dur a terme per la realització del projecte, a més de l'ordre establert per les dependències entre aquestes i un diagrama de Gantt. També s'inclouen els recursos que s'usaran per dur a terme les tasques i es descriuen desviacions possibles i l'efecte que tindran en el projecte.

5.1 Descripció de les tasques

A continuació es detallen totes les tasques del projecte que es duran a terme, en ordre cronològic. Es mencionaran breument els recursos que es faran servir a cada tasca també, però més endavant es detallaran perfectament i es donarà l'estimació temporal de cada tasca.

5.1.1 Gestió del Projecte

Referent a tota la feina d'aprenentatge autònom i lliuraments del mòdul de Gestió de Projectes, aquesta tasca, o assignatura més ben dit, consisteix en l'entrega de 6 lliuraments. La dedicació és de 75 hores en total, separades d'aquesta manera:

- **Abast del projecte i contextualització** 24,50 hores.
- **Planificació temporal:** 8,25 hores.
- **Gestió econòmica i sostenibilitat:** 9,25 hores.
- **Presentació preliminar:** 6,25 hores.
- **Mòdul d'enginyeria de computadors:** 8,5 hores.
- **Presentació oral i document final:** 18,25 hores.

Per la realització d'aquesta tasca es farà ús dels recursos següents: un ordinador, l'aplicació *ShareLatex*.com[17], *Dropbox*[18], el cercador *Google*, eines de *Microsoft Office*, *Gantter*[19] i un mòbil amb càmera.

5.1.2 Estudi del model de programació i el runtime

Abans de començar a dur a terme el projecte, cal estar familiaritzat amb el model de programació que s'usarà i el runtime amb el qual es treballarà. Tal com s'ha descrit en anteriors seccions doncs, el desenvolupador haurà de familiaritzar-se amb el model de programació OmpSs i el runtime Nanos6. Per a dur a terme aquest estudi, el desenvolupador estudiarà amb detall els manuals disponibles i tota la documentació possible referent a ambdós, i serà proporcionat de jocs de proves per executar per veure el potencial i les funcionalitats existents.

Ja que aquesta tasca i les següents no es poden dividir en d'altres, la valoració temporal es farà posteriorment en altres seccions. Pel que fa a recursos per aquesta tasca, es farà servir un ordinador per visualitzar tots els manuals, compilar i executar tots els jocs de proves.

5.1.3 Anàlisi general

Un cop fet l'estudi del projecte (tasca anterior) vindrà la tasca d'analitzar el projecte per definir els objectius i requeriments principals. Aquesta tasca vindrà al seu torn definida per anteriors i posteriors reunions amb el director i co-director del projecte.

Bàsicament es decidirà amb quines funcionalitats extendre el runtime i els mètodes per a dur a terme l'extensió, de manera general i sense entrar molt en detall, per tal que el desenvolupador posi en pràctica els seus coneixements.

Per la realització d'aquesta tasca es necessita un ordinador amb connexió a internet, els manuals mencionats amb anterioritat, el cercador *Google* i *Microsoft OneNote* per tal de portar un seguiment de les reunions i apuntar conceptes importants parlats en aquestes.

5.1.4 Creació de l'entorn

Aquesta tasca té com a dependència la tasca de l'Estudi del model de programació i el runtime i es basa a preparar l'entorn de desenvolupament amb el què es realitzarà el projecte. Aquesta tasca inclou la descàrrega i instal·lació de tot el software necessari per al projecte: el runtime, el compilador, paquets necessaris i Git, a part de configurar el sistema operatiu usat al gust del desenvolupador. Aquesta tasca pot comportar un temps important, ja que tot s'installarà en un sistema operatiu recent i pot ser que alguns paquets no tinguin suport encara.

Per aquesta tasca s'usarà un ordinador portàtil amb doble boot de sistema operatiu: Windows 10 i OpenSUSE Leap, amb connexió a internet. S'usarà el model de programació OmpSs, el runtime Nanos6, el compilador Mercurium, el sistema de control de versions Git i d'altres eines per depurar i editar com, per exemple, *GDB, valgrind, Kate i vim.*

5.1.5 Disseny i implementació

Aquesta tasca depèn de la creació de l'entorn i l'anàlisi general del projecte. Dins d'aquesta tasca es concentren bastants sub-tasques, ja que seria inviable intentar implementar tot el necessari a la vegada, així que s'estableixen dependències. A continuació es llisten les divisions d'aquesta tasca:

1. **Mòdul de mesura:** Té com a objectiu crear un mòdul que permeti obtenir dades a mode de profiling sobre les tasques en temps d'execució, per proporcionar-les al mòdul de predicció.

2. **Clàusula cost:** Aquesta tasca se centrarà a saber quina informació proporcionarà el programador usuari al fer servir aquesta clàusula, i com obtenir aquesta informació per a proporcionar-la al mòdul de predicció. No té com a dependència la tasca anterior, ja que no necessita informació sobre el rendiment per cap raó.

3. **Mòdul de predicció:** Atès que abans de tenir el mòdul de mesura no se sap les dades a les quals es tindran accés, ni com s'hi accedirà, no es podrà implementar el mòdul de predicció fins a haver acabat la implementació del mòdul de mesura i la clàusula cost. Aquesta tasca se centrarà a implementar un mòdul que tingui un sistema de prediccions que proporcioni informació important per, més endavant, prendre decisions en el runtime sobre l'execució.

4. **Mòdul Autofinal:** Aquest serà el mòdul més important que s'implementarà i que farà servir tots els anteriors per tal d'obtenir rendiments competitius. Donat que no és un concepte trivial, s'explicarà més endavant que és exactament aquest mòdul, en la secció 9.4.2.

5. **Modificació en la política d'scheduling:** Després d'implementar totes les tasques anteriors i com a treball futur, es podrà crear un scheduler que aprofiti les dades proveïdes pel mòdul de predicció.

Per a realitzar totes les tasques llistades, es faran servir els recursos esmentats en la secció 5.1.4.

5.1.6 Avaluació del rendiment

Un cop finalitzades totes les tasques esmentades amb anterioritat, es farà una anàlisi dels resultats obtinguts i el rendiment assolit, tal com s'especifica en la secció 4.4.

5.1.7 Memòria final

Com a tasca final es redactarà la memòria final d'aquest projecte i es prepararà tot el material necessari per a la defensa davant del tribunal.

Per dur a terme aquesta tasca s'usarà l'ordinador ja mencionat en tasques anteriors, l'aplicació *ShareLatex.com* i el cercador *Google*.

5.2 Dependències entre tasques

Tasca	**Tasca predecessora**
Gestió del projecte	-
Estudi del model de programació i el runtime	Gestió del projecte
Anàlisi general	Estudi del model de programació i el runtime
Creació de l'entorn	Estudi del model de programació i el runtime
Disseny i implementació	Creació de l'entorn + Anàlisi general
Mòdul de mesura	Creació de l'entorn + Anàlisi general
Clàusula cost	Creació de l'entorn + Anàlisi general
Mòdul de predicció	Mòdul de mesura + Clàusula cost
Mòdul Autofinal	Mòdul de predicció
Avaluació del rendiment	Mòdul Autofinal
Memòria final	Avaluació del rendiment

Taula 1: *Dependències entre les tasques del projecte.*

5.3 Previsió temporal

En la següent taula es mostren totes les tasques amb la seva previsió temporal. Més endavant, en la taula i diagrama de Gantt, es detallarà com es divideix aquesta planificació entre conceptes com implementació, disseny i testeig.

Tasca	**Dedicació (hores)**
Gestió del projecte	75
Estudi del model de programació i el runtime	20
Anàlisi general	50
Creació de l'entorn	20
Mòdul de mesura	130
Clàusula cost	15
Mòdul de predicció	120
Mòdul Autofinal	50
Avaluació del rendiment	15
Memòria final	60
Total	**555**

Taula 2: *Previsió temporal de les tasques del projecte.*

Tal com es veurà més endavant en el diagrama de gantt, cada dia equival a 5.3 hores de treball, pel que les 480 hores de TFG (les que no són de GEP) s'hauran de fer en 98 dies, del 24 d'Octubre de 2016 al 16 de Gener de 2017 més concretament.

5.4 Recursos

A continuació es detallen diferents tipus de recursos necessaris per realitzar el projecte.

5.4.1 Recursos Hardware

- **Ordinador portàtil proporcionat per BSC:** Dell Latitude 14 7000 Series (E7450)[20] amb Intel® Core™ i7-5600U (Dual Core, 2.6GHz, 4M cache, 15W) i 8 GB de RAM.
- **Pantalla proporcionada per BSC:** Per tal de poder treballar amb millors condicions, es proporciona al desenvolupador un monitor Dell Professional P2715Q de 27"[21] al que s'hi podrà connectar el portàtil per tal d'emular un ordinador desktop, amb ratolí i teclats proporcionats també per BSC.

- **Supercomputador *MareNostrum III*:** Proporciona un rendiment de pic de 1.1 Peta-FLOPS, gràcies als 48,896 nuclis Intel® SandyBridge-EP E5-2670 repartits en un total de 3,056 nodes. També disposa de 42 nodes heterogenis amb un total de 84 acceleradors Xeon Phi 5110P. Inclou 115.5 Terabytes de memòria principal i 2 Petabytes d'emmagatzematge en disc. Per connectar els nodes usa Infiniband FDR10 i Gigabit Ethernet.

- **Mòbil amb càmera:** Motorola Moto G4 Plus[22] amb càmera de 13 MegaPíxels per tal de gravar el vídeo de la presentació de GEP.

5.4.2 Recursos Software

- **Sistemes Operatius:** El projecte es desenvoluparà en la distribució OpenSUSE Leap 42.1, tanmateix, les entregues del mòdul de GEP, la memòria i la documentació seran generades en *Microsoft Windows 10*.

- **Control de versions:** Es farà ús de Git.

- ***Debuggar*:** Es farà servir *GDB* i *valgrind*.

- **Editors:** S'usaran els editors *vim i Kate*.

- **Planificació:** Per tal de dur a terme la planificació es farà servir *Gantter*.

- **Software del projecte:** El software base que es farà servir serà el model de programació OmpSs, el runtime Nanos6 i el compilador Mercurium.

- **Altres editors i aplicacions:** Es farà ús d'eines de *Microsoft* com *Word, Excel i OneNote, Dropbox i ShareLatex.com* per redactar la memòria.

5.4.3 Recursos Humans

- **Director i co-director:** Seguiment i supervisió en el compliment de tots els objectius del projecte

- **Personal de suport:** Ajudarà al desenvolupador quan calgui i li donaran consells en la implementació

- **Desenvolupador:** Encarregat de tot el projecte.

5.5 Valoració d'alternatives i pla d'acció

Tal com es pot veure en el diagrama de gantt de l'apèndix, les defenses orals dels Treballs de Final de Grau es poden fer, fins molt tard, el dia 27 de Gener de 2017, pel que la memòria del projecte ha de ser lliurada com a molt tard el dia 16 de Gener de 2017, és a dir, una setmana abans. És per això que, 5 dies abans de l'entrega de la memòria, hauria d'estar enllestit tot el que és referent al projecte. Això dóna un petit marge de llibertat.

Tot i haver repartit el temps d'una manera generosa per les tasques de desenvolupament, ja que són les més propenses a sofrir desviacions temporals, aquest marge ens dóna una mica de joc. Les tasques més propenses a sofrir desviacions en concret són les de disseny i implementació del mòdul de mesura i el mòdul de prediccions. És per això que s'ha estimat més temps en la seva finalització, tot i que és molt probable que si no hi ha una gran quantitat de desviacions, es finalitzin abans del temps predit.

Com és visible, en el pitjor cas que alguna desviació faci dedicar més del marge establert (una setmana) en alguna tasca, s'haurà de reorganitzar la planificació i aplicar mesures com, per exemple, simplificar el problema fent la tasca més senzilla o, en un cas extrem, cancel·lar la tasca completament en cas que no sigui crítica, com són el mòdul de mesura o el mòdul de prediccions. En el cas que sigui crítica, s'haurà d'aplicar una simplificació d'altres tasques amb la supervisió del director i/o co-director.

Gràcies a aquestes mesures correctives, el pla d'acció i el seguiment i supervisió constants portats a terme, s'assegura la finalització del projecte en el temps establert.

6 Gestió Econòmica

6.1 Pressupost dels Recursos

En aquesta secció es mostra una taula per cada tipus de recurs amb les seves característiques més importants, com preu, vida útil, amortització i unitats. Un cop inventariats tots els tipus de recursos hardware, software i humans, amb els preus finals calculats, es mostrarà una taula amb costos indirectes relacionats al projecte i, per últim, es revisaran imprevistos i contingències i es calcularà el pressupost total.

Recurs Humà	**Dedicació(hores)**	**Preu/hora(€/h)**	**Cost total(€)**
Director del projecte	32	50.00[23]	1,600.00
Co-director del projecte	32	50.00[23]	1,600.00
Personal de Suport	20	20.00[23]	400.00
Becari desenvolupador	555	-	-
Total	**639**		**3,600.00**

Taula 3: *Costos dels recursos humans.*

En relació als recursos humans, les estimacions de les hores dedicades del director i el co-director han estat calculades de forma general tenint en compte els dies en els quals es farà el projecte (110), les hores per setmana en reunions (1) i les reunions en aquests dies i altres hores extres d'ajuda o reunions. El personal de suport, en canvi, no té una dedicació preestablerta, ja que guiarà i ajudarà al desenvolupador mentre es vagin trobant problemes, per tant, s'estima que entre 20 i 25 hores seria una bona previsió.

Recurs Software	**Preu(€)**	**Unitats**	**Vida útil**	**Amortització(€/h)**
OpenSUSE Leap 42.1	0	1	3 anys	-
Microsoft Windows 10 Home	135.00 [24]	2	3 anys	Pre-instal·lat
Microsoft Office 365 Universitarios	79.00 [25]	2	3 anys	0.02
Model de programació OmpSs	0	1	-	-
Runtime Nanos6	0	1	-	-
Compilador Mercurium	0	1	-	-
Editors de text: *Kate i vim*	0	1	-	-
Eines per depurar: *GDB*	0	1	-	-
GIT	0	1	-	-
Aplicació *ShareLatex*.com	0	1	-	-
Aplicació *Gantter*	0	1	-	-
Dropbox	0	1	-	-
Total	**428.00**			**0.02**

Taula 4: *Costos dels recursos software.*

Recurs Hardware	**Preu(€)**	**Unitats**	**Vida útil**	**Amortització(€/h)**
Dell Latitude 14 7000 Series (E7450)	1,327.25	1	4 anys	0.17
Dell Professional P2715Q	764.00	1	4 anys	0.1
Supercomputador *Marenostrum III*	-	1	-	-
Motorola Moto G4 Plus	277.09 [26]	1	4 anys	0.03
Total	**2,368.34**			**0.3**

Taula 5: *Costos dels recursos hardware.*

Les hores d'ús del supercomputador no són gratis, no obstant això, són dades no accessibles o proporcionables.

6.2 Costos Indirectes

Com que el projecte es realitzarà en les oficines d'un centre, les hores de treball no estan lliures de costos, ja que es farà ús d'energia i mobiliari. Tot i que els costos esmentats són informació no proporcionable pel centre, es pot fer una estimació del que costaria. Aquesta estimació i d'altres costos són visibles en la següent taula.

Cost Indirecte	**Preu per mes(€/mes)**	**Duració (Mesos)**	**Cost Total(€)**
Despeses d'oficina	50.00	5	250.00
Carnet TJove 6 zones	100.00 [27]	5	500.00
Total			**750.00**

Taula 6: *Estimació de costos indirectes.*

6.3 Imprevistos i Contingències

Com que les tasques o etapes que són més propenses a patir desviacions són el disseny i la implementació del mòdul de mesura i el disseny i implementació del mòdul de predicció, aquesta secció se centra a calcular l'impacte monetari que esdevindria a causa de desviacions en aquestes. Ja que el risc és mitjà i alt respectivament, és raonable fixar un percentatge de contingència del 10% i 15%.

Molts dels recursos proporcionats per la realització d'aquest projecte són proveïts pel *BSC*, cosa que fa que el pressupost sigui bastant senzill i tancat. Per aquesta raó, el percentatge de contingència d'aquest projecte s'ha estimat en un 12%, que s'estima adient i no s'espera sobrepassar. Les desviacions en les tasques poden comportar un augment de, com a molt, 3 hores de dedicació per part del director, co-director i personal de suport, el que suposaria un total de *2 x 3 hores x 50 €/h + 3 hores x 20 €/h = 360 €*. Aquesta xifra queda en la seva totalitat inclosa en el 12% de contingència i es reflecteix en la taula de pressupost pel projecte.

6.4 Pressupost Final

En aquesta secció es presenta la taula definitiva de pressupost del projecte, que inclou totes les previsions per cada tipus de recurs i tasca, calculant el total efectiu del projecte. Cal mencionar que en la taula no apareix cap recurs gratuït, ja que no té cap impacte econòmic ni d'altra naturalesa per la construcció de la taula.

6.5 Control de Gestió

Com una de les desviacions més freqüents en els projectes informàtics sol ser la durada de les diferents etapes, al llarg del projecte poden sorgir imprevistos. És molt important dur un registre específic de les hores invertides en una tasca en comparació amb les hores estimades en la planificació, i portar un seguiment acurat de les tasques per verificar si s'han assolit els objectius. Una bona pràctica per calcular i evitar desviacions del pressupost establert és la comparació dels recursos estimats amb els recursos consumits reals i el seguiment acurat de tasques amb més risc de desviació que, com s'ha esmentat abans, són les de disseny i implementació, més concretament la tasca del *Mòdul de mesura* i la del *Mòdul de Predicció*.

El motiu de portar un seguiment estricte és força raonable, ja que com més aviat es detectin desviacions, abans es podrà actuar per poder corregir-les, minimitzant l'impacte al llarg del temps. Aquest seguiment s'assolirà sense cap dubte, ja que setmanalment hi haurà una reunió de seguiment i, a part d'aquesta, també hi haurà diferents *checks* al llarg de la setmana, on el director i/o el co-director romandran en contacte en tot moment amb el desenvolupador via mail i client *IRC*[28] del centre. També, tal com s'ha proposat abans, l'eina *Microsoft OneNote* es farà servir com a mètode de llista *TO-DO*, on s'hi podran afegir llistes amb *checkmarks* per tal de portar un seguiment de les tasques a fer i les concloses.

Tot i les dificultats que puguin aparèixer, aquest projecte no necessitarà recursos molt diferents dels que s'ha previst, ja que la probabilitat de l'ús d'algun altre recurs hardware no esmentat amb anterioritat és quasi inexistent. Tanmateix, és possible que s'hagi d'usar algun recurs software no esmentat amb anterioritat, el que no hauria de comportar cap problema ni augment de pressupost, com que en l'actualitat hi ha versions gratuïtes de molta varietat de software existent i, a més, el centre podria proveir-les. En el cas d'haver-hi desviacions en els recursos humans, com bé s'ha esmentat abans, la contingència serà capaç d'assolir el cost monetari que impliquin aquestes desviacions. És per aquesta raó que s'ha afegit un 12% de contingència al projecte, per cobrir els possibles riscos existents tot i la prevenció duta a terme. Les desviacions esmentades es calcularan amb aquests indicadors:

- **Desviacions per conceptes:**
  - Desviacions en la realització de tasques en cost = (cost estimat - cost real) * consum d'hores real.
  - Desviacions en la realització de tasques en hores = (consum estimat – consum real) * cost real.
  -Desviacions en recursos en cost = (cost estimat - cost real) * cost real.

- **Desviacions totals:**
  - Desviacions totals en la realització de tasques = (cost estimat total – cost real total).
  - Desviacions totals de recursos = cost estimat total recursos - cost real total recursos.
  - Desviacions totals de costos fixes = cost estimat total fix - cost real total fix.

	Unitats	Amortització(€/h)	Temps(hores)	Total(€)
Costos directes				**167.62**
Gestió del projecte			**75**	**25.5**
Dell Latitude E7450	1	0.17		12.75
Dell Professional P2715Q	1	0.1		7.5
Motorola Moto G4 Plus	1	0.03		2.25
Microsoft Office 365	2	0.02		3
Estudi del model			**20**	**5.4**
Dell Latitude E7450	1	0.17		3.4
Dell Professional P2715Q	1	0.1		2
MareNostrum III	1	-		-
Anàlisi general			**50**	**13.5**
Dell Latitude E7450	1	0.17		8.5
Dell Professional P2715Q	1	0.1		5
Creació de l'entorn			**10**	**3.1**
Dell Latitude E7450	1	0.17		1.7
Dell Professional P2715Q	1	0.1		1
MareNostrum III	1	-		-
Microsoft Office 365	2	0.02		0.4
Disseny i desenvolupament			**330**	**97.47**
Dell Latitude E7450	1	0.17		56.1
Dell Professional P2715Q	1	0.1		33
MareNostrum III	1	-		-
Imprevistos				*8.37*
Mòdul de mesura		10% del cost de la tasca: 35.1€		3.51
Mòdul de predicció		15% del cost de la tasca: 32.4€		4.86
Avaluació del rendiment			**15**	**4.05**
Dell Latitude E7450	1	0.17		2.55
Dell Professional P2715Q	1	0.1		1.5
MareNostrum III	1	-		-
Memòria final			**60**	**18.6**
Dell Latitude E7450	1	0.17		10.2
Dell Professional P2715Q	1	0.1		6
Microsoft Office 365	2	0.02		2.4
Costos indirectes				**750.00**
Costos de recursos humans				**3,600.00**
Total acumulat				**4,517.62**
Contingència	**12%**	Aplicat a: **4,517.62€**		**542.11**
Total sense *IVA*				**5,059.73**
Total amb *IVA*	**21%**	Aplicat a: **5,059.73€**		**6,122.27**

Taula 7: *Pressupost total del projecte.*

7 Sostenibilitat i Compromís Social

7.1 Matriu de Sostenibilitat

En la següent taula es mostra la matriu de sostenibilitat d'aquest projecte.

	Projecte en Producció	**Vida Útil**	**Riscos**
Ambiental	Consum del disseny	Empremta ecològica	Riscos ambientals
	7:10	*15:20*	*0:0*
Econòmic	Factura	Pla de viabilitat	Riscos econòmics
	9:10	*17:20*	*0:0*
Social	Impacte personal	Impacte social	Riscos socials
	9:10	*11:20*	*0:0*
Valoració total	*25:30*	*43:60*	*0:0*
	68:90		

Taula 8: *Matriu de sostenibilitat del TFG.*

7.2 Dimensió Econòmica

Aquest projecte inclou una avaluació de tots els costos dels recursos per la seva realització, siguin materials o humans. Aquesta avaluació, visible en la secció 6, té en compte tots els imprevistos de les tasques del projecte, esmentant plans d'actuació contra aquests.

És un projecte fet en col·laboració amb el Barcelona Supercomputing Center. En termes de competitivitat, no hi té gaire relació, ja que és un projecte pensat internament per software desenvolupat pel centre, és a dir, és una extensió per a millorar un software en desenvolupament per la comunitat científica amb relació al centre.

Tot i que comportaria un major cost de recursos humans, la realització d'aquest projecte es podria dur a terme amb menys temps, ja que qualsevol desenvolupador amb més experiència en el camp del projecte podria realitzar-lo de forma més ràpida i hàbil. No obstant això, seria inviable realitzar-lo amb menys recursos, com que, a part del supercomputador *MareNostrum III*, només es fa ús de recursos bàsics i, tot i així, el supercomputador és necessari per executar aplicacions amb gran capacitat computacional.

La dedicació de cada tasca és proporcional a la seva importància, ja que no s'ha pogut reutilitzar tecnologies o implementacions existents, com ja s'ha mencionat en anteriors seccions.

7.3 Dimensió Social

Espanya ha sofert un lleu creixement econòmic, ja que venia d'una crisi econòmica bastant llarga. Tot i així, l'economia d'Espanya encara està en mínims i la qualitat de vida és molt millorable ara mateix. En l'aspecte polític i regional, Catalunya ha iniciat un procés de ruptura d'Espanya[29], tot i que per ara només s'ha dut a terme una votació no reconeguda per l'estat. És per això que en l'aspecte polític, Catalunya té bastants problemes. Tot i així, el sector informàtic no ha sofert d'aquestes dues crisis, ja que es nodreix de les connexions i els mercats internacionals, el que explica el creixement del sector independentment de la situació econòmica, política i social.

Aquest projecte no té la intenció i és quasi improbable que repercuteixi en afavorir o empitjorar la situació del país. Ja que té la intenció d'afavorir el rendiment d'execucions d'aplicacions paral·leles i, per tant, estalviar recursos als qui treballin amb ell, aquest projecte millorarà la qualitat de vida dels consumidors, ja que permetrà obtenir resultats amb millor rendiment.

En l'àmbit global pot ser que aquest projecte no canviï absolutament res, tanmateix, hi ha una necessitat existent dins del centre, ja que es necessita una infraestructura interna que ajudi a millorar les execucions. A escala competitiva, com s'ha mencionat abans, no hi té cap relació aquest projecte, pel que crec impossible que perjudiqui cap sector, col·lectiu i/o individu.

7.4 Dimensió Ambiental

Com és visible en la taula de pressuposts finals, els recursos necessaris no es poden reaprofitar per altres projectes, a part del portàtil i la pantalla mencionats que són reaprofitats per altres estudiants visitants del centre. El *MareNostrum III*, es podria dir que és reaprofitat, però en realitat és compartit ja que hi han centenars de projectes treballant amb ell. Aquest recurs té un consum molt elevat, ja que és un supercomputador. Tanmateix, tot i no tenir accés a aquestes dades, resultaria difícil calcular el consum d'aquest projecte en relació amb el supercomputador, ja que s'executen centenars d'aplicacions en paral·lel.

En relació amb el consum, l'impacte produït pel portàtil i la pantalla és mínim si comparem amb el consum del supercomputador. Si aquest projecte es donés a terme com a simple treball, sense la necessitat de ser un projecte de final de grau, s'estalviarien recursos humans, ja que el director i el co-director no haurien de guiar en el desenvolupador pel que fa a aspectes tècnics del projecte. També s'estalviaria una gran quantitat de paper i impressions, ja que és probable que la documentació tingui una llargada considerable i s'hagi d'imprimir per al jurat de la defensa.

El desmantellament del projecte, si es volgués dur a terme, seria tan fàcil com tornar els recursos dels quals es fa servei al centre i eliminar la *branca* o *versió de Git* corresponent al projecte. La contaminació del projecte doncs, dependrà de com generi energia el proveïdor, per exemple, d'electricitat.

Pel que fa al projecte, si aconsegueix millorar el rendiment d'aplicacions, aquestes s'executaran més ràpid, el que suposarà menys energia elèctrica gastada i, per tant, una disminució de l'empremta ecològica. Recursos com el portàtil, els sistemes operatius i la pantalla es podran reutilitzar per altres estudiants visitants del centre un cop finalitzat el projecte.

8 Nanos6

Nanos6 és una llibreria runtime dissenyada per servir com suport runtime per entorns paral·lels. Majoritàriament s'usa amb el model de programació OmpSs que és una extensió del model de programació OpenMP basat només en tasques. Aquest runtime es desenvolupa al Barcelona Supercomputing Center, més concretament el grup Programming Models d'aquest centre. A part d'OmpSs, Nanos6 també suporta moltes de les característiques d'OpenMP 3.1 i inclou algunes extensions addicionals (algunes d'elles introduïdes a posteriors releases d'OpenMp).

El runtime proveeix diferents serveis per a suportar el paral·lelisme de tasques usant mecanismes de sincronització basats en dependències. El paral·lelisme de dades també és suportar en el que es refereix a serveis mapejats sobre el seu suport a tasques. Aquestes tasques s'implementen com *User-Level Threads* quan és possible (actualment x86, x86-64, ia64, arm, ppc32 i ppc64 tenen suport). També proveeix suport per mantenir coherència entre diferents espais d'adreces, com pot ser amb GPUs o nodes de clúster, a través d'un mecanisme de directori/cache.

Com a propòsit general, Nanos6 pretén ser usat per recerca en entorns de programació paral·lel. Nanos6 es dissenya per ser extensible a través de plugins. Aquesta extensibilitat no és una característica utòpica, ja que comporta *overheads* que, tot i així, haurien de ser suficientment insignificants per tenir resultats desitjables.

8.1 Cicle de vida del Runtime

Dins de l'execució d'una aplicació d'OmpSs, el runtime té un cicle de vida on s'inicia, s'espera que tota l'aplicació acabi i llavors finalitza. Aquest procés és relativament complex donat que realment és el runtime que inicia l'execució de l'aplicació i no al revés. Primer de tot, el loader carrega la llibreria del runtime i un cop carregada correctament, crida a la funció `nanos_preinit`.

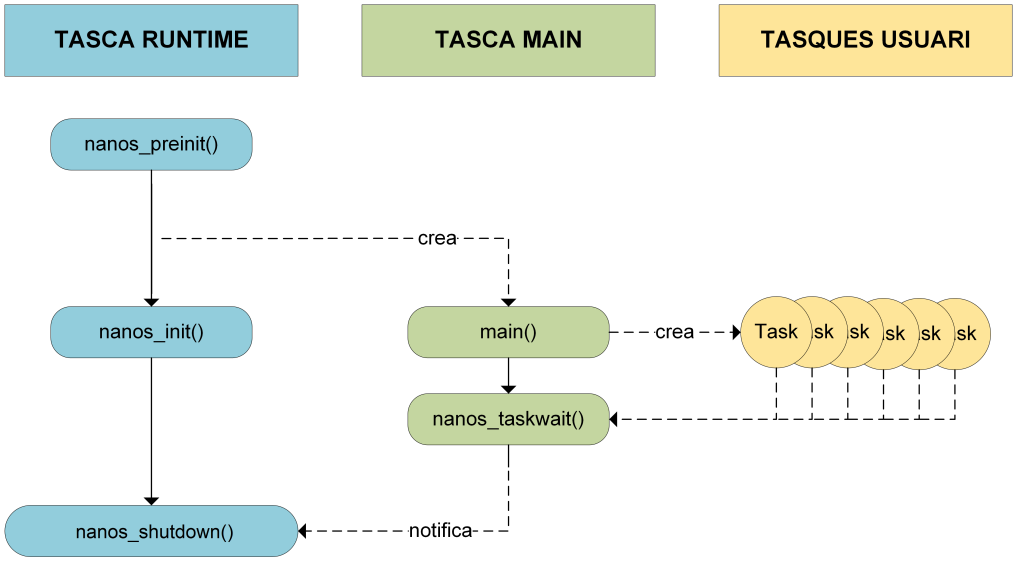

Figura 4: *Dibuix del cicle de vida del runtime.*

Aquesta funció inicialitza totes les estructures internes del runtime i inicialitza els threads i els *schedulers* per tal de poder començar a crear i executar tasques. A partir d'ara, el thread que inicialitza i finalitza les estructures del runtime s'anomenarà tasca runtime, tot i no haver sigut creada explícitament.

Un cop el runtime està llest, el loader crea una tasca anomenada `main` que encapsula una funció especial, explicada en el següent paràgraf. Un cop creada, crida la funció `nanos_init` que en aquest cas no duu a terme cap acció. Llavors, aquesta tasca runtime crida la funció `nanos_shutdown`, la qual espera a què la tasca `main` acabi i un cop acabada, allibera totes les estructures internes, i finalitza el runtime.

La tasca main té assignada una funció que primer de tot, executa la funció `main` del codi d'usuari, llavors espera a les tasques filles que ha anat creant durant l'execució amb un *taskwait*.

En la figura 4 es pot observar de forma gràfica el cicle de vida de la tasca runtime i de la main, explicat anteriorment.

8.2 Clàusules Suportades

Com s'ha mencionat en seccions anteriors l'ús d'OmpSs es basa a usar clàusules dins de pragmas de forma semblant a OpenMP. En aquesta secció s'enumeren i s'explica l'ús de les clàusules usades per aquest projecte i les que hi tenen més relació, deixant l'explicació de la clàusula cost per més endavant, ja que abans d'aquest projecte no es feia usar per res i és en aquest projecte on pren sentit.

- `task`: Especifica que el bloc de codi envoltat pel pragma es tractarà com una nova tasca o unitat de treball a executar per un thread.

- `taskwait`: Mètode de sincronització que pausa l'execució fins que totes les tasques en la visibilitat del pragma que conté el taskwait s'han acabat d'executar.

- `firstprivate, private & shared`: La clàusula `private` declara les variables de la seva llista com privades per cada thread, la clàusula `shared` les declara com compartides per tots els threads i la clàusula `firstprivate` especifica que cada thread tindrà la seva pròpia instància de la variable i que s'inicialitzaran totes amb el valor que tenia la variable abans d'arribar al pragma.

- `in, out & inout`: Declara dependències d'entrada, sortida o ambdues per la tasca del pragma. La clàusula d'entrada o `in` especifica que la tasca generada serà dependent de totes les tasques prèviament generades que referencien almenys un dels objectes de la dependència amb tipus `out` o `inout`. Les clàusules de sortida i entrada-sortida especifiquen que la tasca generada serà dependent de totes les tasques prèviament generades que referencien almenys un dels objectes de la dependència amb tipus `in`, `out` o `inout`. En la figura 5 es pot visualitzar l'ús d'aquestes clàusules.

```c
int main () {
    int a = 0;

    #pragma oss task in(a)
    printf("Ini %d\n", a);

    #pragma oss task inout(a)
    a = 1;

    #pragma oss task in(a)
    printf("Fin %d\n", a);

    #pragma oss taskwait
}
```

```c
int main () {
    int b[2]; b[0] = 0; b[1] = 0;

    #pragma oss task in(b[0; 2])
    printf("Ini %d\n", b[0] + b[1]);

    #pragma oss task inout(b[1])
    v[1] = 1;

    #pragma oss task in(b[0; 2])
    printf("Fin %d\n", b[0] + b[1]);

    #pragma oss taskwait
}
```

Figura 5: *Codis d'exemple amb dependències.*

- `label`: Serveix com a propòsit per donar nom a una instància de tasca. Pel que fa a aquest projecte i com bé s'explicarà més endavant, aquesta clàusula ajuda a identificar quines tasques haurien de compartir certa informació de timing.

- `weakin, weakout & weakinout`: També usades per declarar dependències d'entrada, sortida o ambdues, aquestes clàusules tenen la diferència que, en comptes d'especificar que la tasca generada té dependències, especifiquen que alguna de les tasques generades per la tasca en qüestió, és a dir, alguna de les tasques filles, tenen la dependència i, per tant, la tasca que s'està generant pot començar a executar-se.

- `cost`: Aquesta clàusula és la més important en relació al projecte i serveix per donar una `hint` al runtime del cost algorítmic que té una tasca. A través d'aquesta hint i d'altres mètriques obtingudes pel mòdul de mesura, es poden normalitzar resultats per fer prediccions, cosa que s'explicarà més endavant en altres seccions.

9 Disseny i Implementació

Aquesta secció té com a objectiu donar a conèixer la part més tècnica d'aquest projecte, explicant cada part del disseny i la implementació en detall. Les explicacions llavors, vénen donades en ordre cronològic d'implementació. Per a aclarir terminologia sobre els mòduls, es presenta tot seguit un diagrama que mostra com es relacionen els mòduls implementats entre ells i, més endavant, s'explica en detall tot el que està relacionat amb cada mòdul.

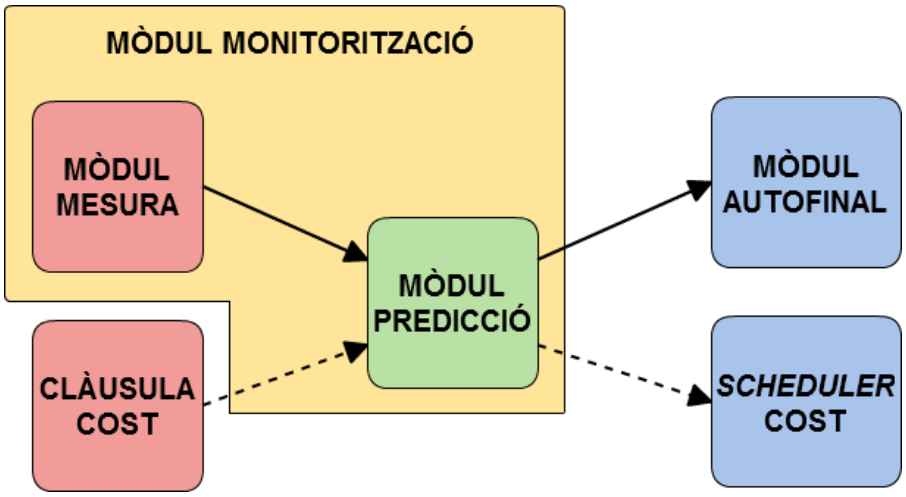

Figura 6: *Esquema conceptual de la relació entre els mòduls implementats.*

Com és visible, tot i que el mòdul de predicció pot usar informació de la clàusula cost, és possible que no pugui fer-ho (d'aquí la raó de per què la relació es visualitza com una recta discontinua). A més, com s'explicarà més endavant, tot i que per aquest projecte s'ha fet servir el mòdul de predicció per a relacionar-lo amb el mòdul autofinal, també es podrien implementar schedulers que utilitzessin la informació del mòdul de predicció. No obstant això, com no s'ha implementat cap scheduler, la relació també es deixa com discontinua per donar a entendre que es podria arribar a fer com a treball futur.

9.1 Mòduls de Monitoratge: Mesura i Predicció

Abans de començar amb el mòdul de mesura, és a dir, el mòdul del runtime que s'encarrega de capturar informació de l'execució d'aplicacions, es van explorar altres opcions. La més clara era començar des de zero a implementar un mòdul lleuger, tanmateix, es va analitzar l'opció d'adaptar una infraestructura ja existent, la d'instrumentació.

9.1.1 Instrumentació

En la implementació actual de Nanos6, existia també un mòdul d'instrumentació. Aquest mòdul permetia, a través de la variable d'entorn ***NANOS6***, obtenir diferent informació, part de la qual era necessària per a aquest projecte. En concret, es necessitava alguna manera per mesurar, per cada tipus de tasca, el seu temps d'**execució d'usuari**, el temps d'execució en **mode runtime** (p.ex. creant altres tasques), el temps que estava **bloquejada** per dependències o taskwait i el temps que havia estat **ready** abans de ser executada.

En la següent taula es poden veure els diferents tipus de mòduls suportats pel runtime, cada un oferint una sortida diferent segons les necessitats del programador.

VALOR	**CONFIGURACIÓ DEL RUNTIME**
optimized	Valor per defecte, el runtime estàndard basat en pthreads.
argobots	Implementació del runtime usant/basada en argobots.
debug	Runtime compilat sense optimitzacions i assercions activades.
verbose	Runtime instrumentat per emetre un log de l'execució
verbose-debug	Configuració verbose + debug
graph	Instrumentat per produir un graph de l'execució.
profile	Instrumentat per produir profiling a nivell de funció i codi font.
stats	**Instrumentat per produir un resum de mètriques d'execució.**
null	Runtime dummy que executa tot el codi seqüencialment.

Taula 9: *Diferents valors de la variable d'entorn NANOS6 i la configuració establerta*

Així doncs, un cop estudiada aquesta solució es veu que, el runtime amb la variable *stats* proporciona les mètriques d'execució llistades a continuació, de les quals només ens són d'utilitat les ressaltades.

- **Nombre d'instàncies d'una mateixa tasca**
- **Temps d'instanciació mitjà**
- **Temps ready mitjà**
- **Temps ready (sense dependències) mitjà**
- **Temps d'execució de codi usuari mitjà**
- **Temps en estat bloquejat mitjà**
- Temps zombie mitjà
- Temps de vida mitjà d'una tasca
- **Número de CPUs**
- Número total de threads
- Mitjana de threads per CPU
- Mitjana de tasques per thread
- Mitjana de temps de vida d'un thread
- Mitjana de temps útil d'un thread

Després de fer algunes proves amb el runtime i la instrumentació d'aquest, es va veure que aquest mòdul d'instrumentació es podria adaptar per, en comptes de fer la funció que feia que és obtenir totes les mètriques llistades anteriorment en finalitzar l'execució d'una aplicació, poder obtenir aquesta informació de manera interna, és a dir, proporcionant les mètriques a altres mòduls del runtime implementats en un futur. Tanmateix, aquesta decisió d'adaptar el mòdul d'instrumentació es va descartar relativament de pressa, ja que aquest mòdul estava pensat per donar un llistat de mètriques *post-mortem*, és a dir, un cop l'execució d'una aplicació havia finalitzat. Aquest propòsit no servia per a l'objectiu d'aquest projecte, ja que es necessitava un mòdul lleuger i disponible en temps real, és a dir, tenir la informació en temps d'execució per tal de prendre decisions en el runtime i millorar l'execució d'aplicacions.

Per aquesta raó, es va decidir dissenyar i implementar un mòdul nou i no reutilitzar la instrumentació existent.

9.1.2 Obtenció de les Mètriques

Donat que un dels requisits més importants del mòdul era el de ser lleuger, abans d'escollir qualsevol mètode per analitzar el temps d'execució, es va haver de fer una anàlisi de les opcions existents i els pros i contres que venien amb elles. A continuació es donen a conèixer les possibles opcions, l'escollida i el perquè de l'elecció.

- **C - <sys/times.h>:** Aquesta llibreria proporciona un ús bastant senzill, amb el simple fet d'instanciar estructures de tipus `clock_t` que contenen cronòmetres i l'ús de la crida `times(clock_t)`, ja es pot fer servir. Pel que fa a l'afinitat, dóna una estimació del temps d'usuari i crides a sistema fins als mil·lisegons.

- **C - <ctime>:** Amb unes simples crides `std::clock()` per començar i parar els cronòmetres, aquesta opció dóna una estimació del temps d'execució en segons pel que, tot i que l'*overhead* afegit no era excessiu, l'afinitat va ser el causant de descartar aquesta opció.

- **C - <sys/time.h>:** Considerant que aquesta opció no només mesura el temps entre dos punts d'un programa sinó que també mesura d'altres mètriques no importants per l'objectiu d'aquest projecte i que, a més, afegeix un overhead important a l'execució d'aplicacions, aquesta opció també es va descartar.

  L'ús consistia a preparar estructures amb moltes mètriques de memòria, I/O, CPU i threads i emplenar la informació d'aquestes estructures amb crides a `getrusage()`. Pel que fa a l'afinitat, és de fins a microsegons.

- **C - <time.h>:** També descartada per mesurar masses mètriques i per tant afegir overhead considerable, aquesta opció té com afinitat fins a nanosegons, característica desitjada per l'objectiu del projecte. Similar a altres opcions, l'ús és relativament senzill i consisteix en crides a `clock_gettime(id, struct)` on *id* fa referència al tipus de clock a usar i *struct* és una estructura simple que es fa servir com a fita, pel que se'n necessiten dues (principi + final).

- **C++11 - <chrono>:** L'ús d'aquesta llibreria no és senzill pel fet d'haver d'usar bastants casts i funcions. Gràcies a això però, no hi ha gaire overhead ja que cada mètrica té crides úniques. A part d'això, aquesta llibreria suporta poder calcular diferents temps segons l'afinitat desitjada (nanosegons, microsegons, mil·lisegons i segons).

La solució escollida és usar la llibreria `chrono` de `C++` per obtenir informació de timing de les tasques, ja que posant a prova l'obtenció de la informació fent bastants crides a les diferents funcions de les llibreries mencionades, es va veure que era l'única solució que incrementava el temps d'execució de manera negligible, mentre que totes les altres opcions eren bastant costoses. A part d'això té també només una funció clara per cada mètrica, pel que no hi ha overhead afegit de mesurar altres mètriques quan l'única que es vol obtenir és la de temps d'execució. Pel que fa a afinitat, aquesta solució permet analitzar execucions fins als nanosegons cosa que, com s'ha mencionat anteriorment, era gairebé un requisit.

El fet d'usar aquesta solució deriva que el runtime suporti C++11, cosa que fins ara ja existia, pel que aquesta característica no va suposar cap problema en relació a la portabilitat del runtime. Tot i així, posteriorment va sortir l'impediment que el fitxer `nanos6_rt_interface.h` havia de ser en pur `C`, pel que les estructures no es podien inicialitzar allà, cosa que era necessària, ja que aquest fitxer conté la declaració del `taskinfo`, estructura que es definirà més endavant. Això va tenir solució gràcies al fet que, des del runtime, en temps d'instanciació d'una tasca s'inicialitzen les estructures de timing com es veurà més endavant. Per fer-ho compatible, les estructures de timing es declaren com un punter a void `(void *)` dintre d'una tasca i, posteriorment, es transforma en un punter a les estructures apropiades amb un simple cast fora de la visibilitat d'aquest fitxer.

9.1.3 Estructures del Runtime

El runtime com s'ha mencionat abans es basa en tasques i aquestes, tot i ser instàncies diferents, poden compartir informació. D'aquesta compartició surt el `taskinfo`. Moltes tasques poden compartir aquesta informació però, pel que fa a anàlisis d'execució, només ens interessa tenir una mitjana. D'aquí surt la idea de tenir per cada instància de tasca les seves mètriques de rendiment i un cop finalitzada l'execució de la tasca, actualitzar la mitjana de totes les mètriques analitzades per aquell tipus de tasca o, en altres paraules, actualitzar les mètriques del taskinfo.

Resumint doncs, les modificacions en estructures ja existents del runtime es basaran a afegir estructures de mesura de timing parcial en la definició de l'objecte `Task` i afegir estructures permanents en la definició de l'estructura `nanos_task_info` que contindran una mitjana de les estructures parcials.

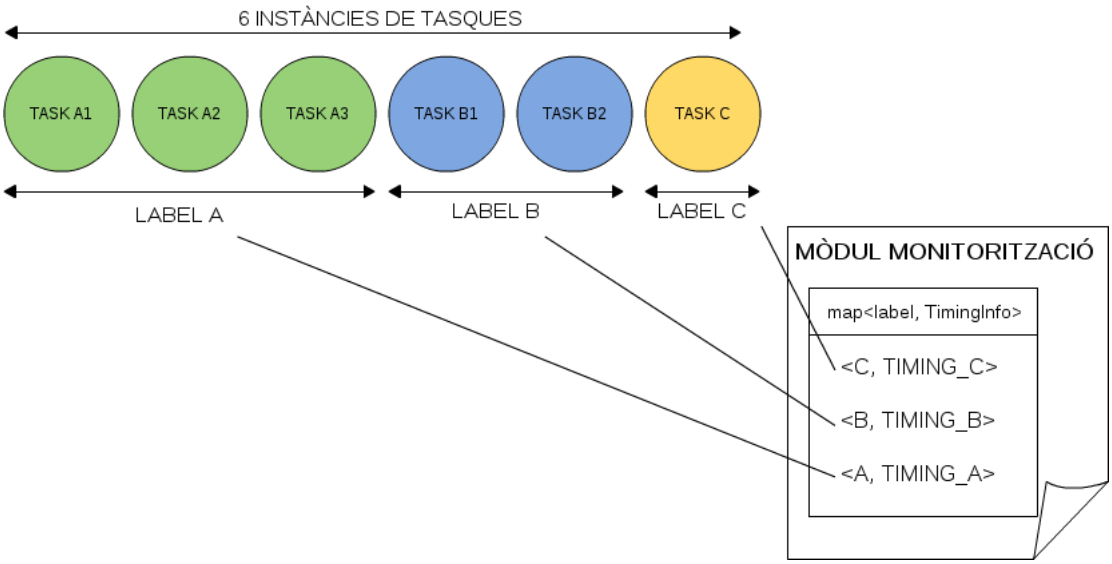

Figura 7: *Relació entre tasques i labels en el mòdul de mesura.*

Aquesta última modificació però no va ser possible ja que `nanos_task_info` és una estructura instanciada pel compilador i, segons la visibilitat de les tasques, era possible que dues tasques que haurien de compartir informació no la compartissin. Aquest problema es va solucionar gràcies a què el mòdul de mesura havia de ser un mòdul que es cridés de forma estàtica. Amb aquesta condició i tenint en compte que podem identificar tasques a través de la clàusula label, es crea una relació entre labels i estructures de timing que haurien d'estar dins el `taskinfo`. Aquesta relació es localitza en un `std::map` dintre del mòdul de mesura.

9.1.4 Estructures de Timing

- **Chrono:** Chrono és l'objecte base de mesura del temps i conté el que és indispensable per a fer funció de cronòmetre. Aquest objecte conté un acumulador on es va afegint el temps d'execució mentre es fa servir el cronòmetre. L'ús és relativament senzill, conté mètodes per començar i parar el cronòmetre en si i mètodes per obtenir la durada capturada per aquest.

- **TaskChronos:** Tal com indica el nom, aquesta classe és la que es localitza dins la classe `Task` i conté tots els cronòmetres necessaris per obtenir la informació de totes les mètriques per una instància de tasca.

  En concret, aquesta classe conté 5 objectes de tipus `Chrono`, un per capturar el temps que una tasca està **ready** que és el temps entre l'instant en què una tasca s'ha acabat de generar i l'instant en què es comença a executar, un altre per capturar el temps d'execució d'usuari que és el temps d'execució del codi de la tasca en si, un tercer per capturar el temps que una tasca està bloquejada en un taskwait, un altre per capturar el temps **pending** o, en altres paraules, el temps d'espera d'una tasca fins que les seves dependències s'han resolt, i un últim `Chrono` per mesurar el temps que una tasca està creant codi de runtime com, per exemple, en el cas de crear altres tasques.

  A més, aquesta classe conté una variable que acumula el temps d'execució acumulat de totes les seves filles i un punter al cronòmetre que està funcionant en un instant donat, és a dir, un punter a l'estat de la tasca.

- **TaskMonitoringInfo:** Aquesta classe és la que conté la informació afegida de totes les mètriques d'un cert tipus de tasca identificat pel label. Conté un comptador de tasques generades per aquell tipus de tasca i un altre comptador per saber quantes tasques d'aquell tipus que almenys tenen una tasca filla s'han instanciat.

  Els dos comptadors s'usen per obtenir una aproximació del nombre de tasques que són generades per aquest tipus de tasca. Si tenim el nombre de tasques generades per un cert tipus de tasca i el nombre de tasques d'aquest mateix tipus que tenen almenys una filla, podem estimar una mitjana de l'`aritat` d'aquell tipus de tasca, és a dir, una estimació del nombre de tasques que es generaran per una certa instància de tasca.

  A més dels dos comptadors mencionats, aquesta classe també conté un acumulador de la llibreria `boost` del tipus `accumulator_set` i un `spinlock` per tal d'obtenir accés atòmic als acumuladors. Aquest set ens permet acumular els costs unitaris d'un cert tipus de tasques i amb una sola crida a `rolling_mean`, obtenir la mitjana del cost unitari amb un interval o finestra de valors, per tal de sempre obtenir la mitjana dels 'N' elements més recents. Més endavant s'explicarà en detall el perquè de totes aquestes variables i com es fan servir.

  Pel que fa a mètodes, aquesta classe s'usa per acumular valors i obtenir una mitjana del cost unitari d'una tasca, pel que té dos mètodes; `void accumulateTimes(Task * task)` que acumula les mètriques d'una tasca en l'acumulador del seu tipus, i `double getUnitaryCostMean()` que retorna la mitjana del cost unitari d'una tasca.

Resumint doncs, cada instància de tasca conté un objecte `TaskChronos` que al seu torn conté objectes de tipus `Chrono`. Les mètriques de cada objecte `TaskChronos`, un cop finalitzada l'execució d'una tasca, s'acumulen en el `TaskMonitoringInfo` del seu tipus de tasca, identificat en el mapa del mòdul de monitoratge pel `label` de la tasca.

9.1.5 Implementació dels Mòduls de Mesura i Predicció

Al llarg d'aquest document s'ha anat mencionant que els mòduls implementats havien de ser totalment independents d'altres classes. Aquesta condició ve de l'objectiu de no haver d'afegir una dependència de classes del runtime cap a les mesures, ja que els mòduls han de treballar independentment del runtime només intervenint quan calgui. Per aquesta raó, aquests mòduls són usats de forma estàtica amb un *singleton*.

```
// Initialization of the monitoring singleton instance
static Monitoring* initialize(bool activation);

// Destructs the monitorization module
static void finalize();

// Initializes timing structures for the newly created task
static inline void taskCreated(Task * task);

// Retrieve a task's type monitoring information by its tasklabel
static TaskMonitoringInfo * getTaskMonitoringInfo(std::string tasklabel);

// Accumulate a task's timing records in the appropiate structures
static void updateCost(Task * task);

// Retrieve the unitary cost of a task.
static double getUnitaryCost(Task * task);

// Start timing for a task on pending state
static inline void startPendingTiming(Task * task)

// Start timing for a task on ready state
static inline void startReadyTiming(Task * task)

// Changes the timing being performed according to the current blocked state
static inline void taskIsBlocked(Task * task)

// Changes the timing being performed according to the current executing state
static inline void taskIsExecuting(Task * task)

// Changes the timing being performed according to the current pending state
static inline void taskIsPending(Task * task)

// Changes the timing being performed according to the current ready state
static inline void taskIsReady(Task * task)

// Changes the timing being performed according to the current runtime state
static inline void taskIsRuntime(Task * task)

// Stop timing for a task
static inline void stopTiming(Task * task)
```

Figura 8: *API del mòdul de mesura.*

Tal com indica el seu nom, el mòdul de mesura és el responsable de gestionar el monitoratge de les tasques, per així poder obtenir les mètriques. Com a tal, és molt semblant a usar una API dins del runtime. En la figura 8 es mostra de forma resumida l'API d'aquest mòdul, amb una breu descripció de cada mètode just abans de la capçalera. Per tal de fer-ho més entenedor, s'han obviat alguns mètodes relacionats amb altres mòduls que s'explicaran més endavant.

El mòdul de predicció fa servir el mòdul de mesura i posteriorment la clàusula cost. Com a tal, es compon de pocs mètodes encarregats de la lògica darrere d'usar el mòdul de mesura i la clàusula cost i, per això, es va decidir implementar els dos mòduls en la mateixa classe estàtica, el mòdul de monitoratge.

9.2 Profiling de Tasques

Com es mostra en el tercer apèndix, el runtime s'ha de modificar per tal de poder fer un profiling de les tasques i saber en cada moment en quin estat està una tasca.

En començar a crear i fer el submit d'una tasca, hi ha dues possibles opcions; en la primera, la tasca no té dependències no resoltes, pel que pot passar directament a l'estat **Ready** on s'hi estarà fins que un thread estigui disponible, mentre que la segona opció és que la tasca tingui dependències no resoltes pel que passarà a l'estat **Pending** fins que les dependències es resolguin. Aquests dos camins d'estats estan marcats en l'apèndix mencionat amb color negre.

Un cop un thread estigui disponible per executar la tasca, aquesta passarà a l'estat **Execution** i passarà a executar codi d'usuari. Arribats a aquest punt, hi poden haver 3 esdeveniments en el cicle d'execució fins que la tasca acabi.

El primer esdeveniment possible, marcat en color porpra, escenifica trobar-se un taskwait en el codi d'usuari, pel que la tasca passarà de l'estat **Execution** a l'estat **Blocked**. Un cop se satisfaci el taskwait, la tasca passarà breument a estar en estat **Runtime** i tot seguit tornarà a executar codi d'usuari, pel que tornarà a l'estat **Execution**.

El segon esdeveniment és que la tasca hagi de crear un altre o altres subtasques. En aquest cas, com es pot veure marcat en vermell, la tasca passarà a l'estat **Runtime** per crear la tasca (`nanos_create_task`) i fer-ne el submit (`nanos_submit_task`). Un cop acabada la generació, es torna a l'estat **Execution**.

El tercer i últim esdeveniment, marcat en blau, és quan la tasca ha acabat la seva execució, pel que es trobarà amb un taskwait implícit. Abans de passar per l'estat **Blocked** però, passarà per **Runtime**. Un cop satisfet el taskwait i alliberades les dependències que pogués tenir, passarà de nou a **Runtime** abans d'actualitzar les estructures de timing amb les mètriques per si ha de fer el submit d'alguna tasca la qual ha de passar a estat Ready per haver alliberat dependències.

9.3 Clàusula Cost

Per tal de fer prediccions es podrien usar les mitjanes capturades pel mòdul de mesura sense cap problema i això és en realitat el que es fa si no tenim cap informació d'aquesta clàusula. Tanmateix, tenir informació proporcionada per aquesta clàusula ens ajuda a normalitzar el cost d'una tasca i, si està executant codi sense cap tipus de funció algorítmica, no tindrà cap efecte negatiu ja que simplement serà una mitjana més però, si per contrari executa alguna funció algorítmica amb una expressió amb cost quadràtic o logarítmic per exemple, ens ajudarà a tenir una mitjana normalitzada i, per tant, molt més verídica. Així doncs, es poden fer prediccions i prendre decisions només amb els mòduls de mesura i predicció, però aquesta clàusula era molt interessant de ser suportada per les raons que s'han esmentat.

Previ a aquest projecte, la clàusula cost no estava suportada pel que es va haver de modificar el model per tal d'afegir la clàusula i el compilador Mercurium per tal de suportar-la. Aquestes adaptacions per tal d'introduir la nova clàusula i suportar-la es van fer respectivament pel personal adequat a cada projecte, és a dir, els encarregats del model de programació OmpSs la van afegir i el personal encarregat del compilador Mercurium va afegir suport per la clàusula. Pel que fa al runtime, tot el que deriva de l'ús de la clàusula es va implementar en aquest projecte.

Aquesta clàusula serveix com a *hint* pel runtime, usada pels programadors que en fan ús. Aquesta hint és informació sobre el pes relatiu d'una tasca o, en altres paraules, el cost algorítmic. En la figura 9 es pot veure l'ús de la clàusula per alguns algoritmes coneguts i a continuació es llisten algunes de les característiques més destacables.

```
#pragma oss task cost(N*N)
void QuickSort(int * const src, int * const dst, const size_t N);

#pragma oss task cost(dim*dim*dim)
void MatMul(int dim, double * const A, double * const B, double * const C);

#pragma oss task cost(N*log(N))
void MergeSort(int * const src, int * const dst, const size_t N);
```

Figura 9: *Benchmarks exemple amb clàusula cost.*

- Per cada tasca, el runtime té una estructura on relaciona el seu label amb les mitjanes dels temps d'execució.
- La clàusula accepta variables, símbols i fins i tot crides a funcions d'altres llibreries gràcies al compilador. Això fa possible l'ús de crides a funcions logarítmiques, exponencials, quadràtiques, etc.
- El cost de cada tasca es normalitza més endavant (p.ex. dividint el temps d'execució pel cost proporcionat amb la clàusula).
- El cost normalitzat o cost unitari fa possible una comparació justa amb diferents mides d'entrada i tipus de tasques.
- Els resultats poden ser usats per prendre decisions en el runtime com es veurà més endavant i estimar amb precisió el temps d'execució d'una aplicació.

- Tasques d'un mateix tipus amb costs similars aporten resultats i prediccions bones, tanmateix, tasques amb costs molt diferents poden comportar-se de manera no esperada. Aquesta possible problemàtica no era interessant per l'objectiu del projecte ja que, donant un exemple justificat, els benchmarks més usats no acostumen a usar mètodes o algoritmes amb costs diferents. Així doncs, aquesta qüestió es comentarà i s'explicarà amb més detall més endavant, en la secció 12.

Fins a la clàusula cost, ja es tenia el mòdul de mesura funcionant, així que es van poder analitzar diferents benchmarks per fer una primera aproximació a les prediccions que hi podrien haver. D'aquestes execucions en va sortir la taula mostrada a continuació.

TASK	**PROBLEM SIZE**	**COST EXPRESSION**	**COST VALUE**	**TIME**	**UNITARY COST (s)**
QuickSort	N = 200.000	*O(N*log(N))*	1.060.206	630,61 μs	0,5948 * 10 ^-15
	N = 1.000.000		6.000.000	3535,2 μs	0,5892 * 10 ^-15
InsertSort	M = 200.000	*O(M*M)*	200.000^2	184,80 ms	4,6202 * 10 ^-12
	M = 1.000.000		1.000.000^2	4604,40 ms	4,6044 * 10 ^-12

Taula 10: *Exemple d'ús del mòdul de mesura amb la clàusula cost*

En la taula es poden veure les característiques obtingudes de l'execució de dos algoritmes senzills amb el runtime funcionant amb el mòdul de mesura complet i la clàusula cost. En la segona columna es pot veure la mida d'entrada de l'algoritme o el que és el mateix, el nombre d'elements a ordenar. La tercera columna fa referència a la clàusula cost i és exactament el que contenia, una expressió algorítmica del cost relatiu de l'algoritme. La quarta columna conté el valor actual rebut pel runtime a través de la clàusula cost un cop el compilador l'ha traduït, la penúltima columna conté el temps d'execució total de l'algoritme que en aquest cas és una tasca sencera i l'última columna conté el cost unitari.

Com s'aprecia en la taula, tenint l'expressió algorítmica i el temps d'execució de les execucions amb mida 200.000, podem normalitzar un cost unitari i, com es veu, el de les execucions amb mida 1.000.000 és gairebé idèntic al cost unitari anterior.

Així doncs, tenint una estimació molt bona del que poden arribar a tardar algunes tasques, podem planejar l'execució per tal de fer un balanceig per exemple, o fins i tot canviar el mètode d'execució d'aquestes tasques les quals podem predir el seu temps d'execució. En les següents seccions s'introdueix l'ús de les prediccions a través de nous mòduls i més endavant es poden veure alguns resultats i algunes comparatives entre versions del runtime.

9.4 Mòdul Autofinal

Un cop funcionant l'obtenció de mètriques i les prediccions, per arribar a l'objectiu d'aquest projecte faltava un mòdul que, usant aquesta informació millorés l'execució de posteriors tasques.

En un principi aquest mòdul era una extensió del mòdul de monitoratge amb mètodes que permetessin a un nou *scheduler*, que s'hauria d'implementar, alimentar-se d'aquesta informació. No obstant això, implementar un scheduler des de zero hauria requerit un gran esforç i una major dedicació de recursos temporals. Per aquests motius, es va decidir aproximar a una altra solució que també s'havia d'implementar i tenia més expectativa i atenció, i que no incloïa de moment cap scheduler. Per entendre aquesta solució però, abans s'ha d'introduir un nou concepte en el runtime, el qual s'explica a continuació.

9.4.1 Clàusula Final

La clàusula final introduïda en el runtime té com a finalitat aplicar un *threshold* manual en tasques. La idea és que, arriba un moment en què crear més tasques suposa un overhead més gran que no pas executar el codi de la tasca seqüencialment, pel fet d'haver de gestionar massa tasques. Així doncs, amb la clàusula final, un programador podria explícitament declarar condicions en el seu codi on, si s'arriba a un màxim nombre de tasques, és a dir un threshold, les noves tasques no es creïn i simplement el mateix thread executi el codi de la tasca.

El funcionament és senzill i s'adapta a les clàusules que s'han vist anteriorment: el compilador, en veure una clàusula final, crea dos tipus de codis, un d'ells on es genera la tasca i un altre on el codi és una còpia sense el pragma. En la figura 10 es pot veure un exemple de l'ús de la clàusula final en el solver de l'algoritme d'*N-Queens*.

```
void solve(int n, const int col, sol_node_t& sol, int& result)
{
    if (col == n) {
        result++;
    }
    else {
        for (int row = 0; row < n; row++) {
            if ( !check_attack(col, row, &sol) ) {
                sol_node_t new_sol;
                new_sol.prev = &sol;
                new_sol.row = row;

                #pragma oss task final(FINALDEPTH <= col) label(RecursiveSolver)
                solve(n, col + 1, new_sol, result);
            }
        }
    }
}
```

Figura 10: *Solver de l'Algoritme NQueens usant Clàusula Final.*

Segons es pot veure en la figura, el programador del solver d'aquest algoritme especifica que si les columnes han arribat a un cert límit, s'han creat massa tasques i, per tant, a partir d'aquell moment les noves tasques seran tractades com a *final* ja que la condició es compleix.

Que una tasca sigui tractada com final significa que, a partir d'ara, aquella tasca serà executada sencera sense generar més subtasques. Això equival a què, si una tasca és final i dins d'ella hi ha un loop que genera 5 tasques, el codi d'aquestes 5 tasques s'executarà com si formés part de la tasca final, sense arribar a generar-les. Aquesta funcionalitat normalment és desitjable per a tasques les quals el seu temps d'execució és menor que el temps d'execució de runtime que generen. En altres paraules, crear, fer el submit, i esperar al fet que totes les dependències estiguin resoltes o alliberar les dependències que calguin, genera més treball que el codi d'usuari dins la tasca en si.

9.4.2 Autofinal

En la secció 9.4.1 s'introdueix la clàusula final especificant que és el programador usuari qui ha de programar aquesta clàusula amb una condició encertada per tal d'obtenir un *threshold* suficientment bo. El fet d'haver-ho de fer explícitament està condicionat per haver de tenir coneixement de l'aplicació, la granularitat de les tasques i si hi ha diferència entre elles. Totes aquestes condicions fan que el programador usuari hagi d'estudiar l'aplicació en qüestió i fins i tot provar diferents thresholds manualment, pel que s'han de portar a terme algunes execucions prèvies per a veure el comportament de l'aplicació.

Totes aquestes característiques fan que sigui interessant tenir alguna forma d'activar la clàusula final automàticament amb un threshold implícit. D'aquí surt la idea de tenir un nou mòdul anomenat automatic final.

Aquest mòdul, amb la informació obtinguda del mòdul de predicció (en part mòdul de monitoratge), és capaç de calcular una estimació del temps d'execució d'una tasca i determinar si és interessant que la tasca pugui ser final. A continuació s'introdueix el pseudocodi de la decisió de si una tasca ha de ser final o no i tot seguit s'explica el codi usat i el perquè de cada part. Més endavant es mostren les modificacions a mòduls anteriors i al runtime a causa d'aquest codi.

```
Function is_automatically_final():
    arity = children_tasks / parent_tasks;
    if no_cost_available then
        return (arity ^ recursive_depth) > (CPUS * TASKS_PER_CPU);
    end
    else
        return (cost * unitary_cost) < THRESHOLD;
    end
end
```

Algorithm 1: *Pseudocodi de les condicions perquè una tasca sigui considerada final.*

```
bool Monitoring::isAutomaticallyFinal(Task * task)
{
 if (AutoFinal::isEnabled() && !task->hasWeakDepend()) {
    std::string taskLabel = task->getLabel();
    double unitaryCost = getUnitaryCost(task);

    TaskMonitoringInfo * info = getTaskMonitoringInfo(taskLabel);
    double arity = info->_childrenTasks) / info->_parentTasks;

    // 1. No cost information for this type of task.
    if (unitaryCost < 0) {
       return (pow(arity, ((double) task->getRecursiveDepth())) >=
           _monitor->_totalCPUs * AutoFinal::getTasksPerCPU());
    }

    // 2. There's cost information for this type of task.
    else {
       unsigned long long cost = task->get_cost();
       return ((cost * unitaryCost) < AutoFinal::getThreshold());
    }
 }
 return 0;
}
```

Figura 11: *Codi actual de la decisió sobre si una tasca és final o no.*

Com és visible a la figura 11, hi han dues parts ben diferenciades. La primera té relació amb la no disponibilitat d'informació de timing. Hi ha dos escenaris els quals poden provocar aquesta situació, el primer és quan una tasca és la primera a ser executada i per tant encara no s'han actualitzat les estructures de timing i el segon pot ocórrer quan s'ha entrat en recursivitat. Per entendre millor el segon escenari es pot pensar en algoritmes com mergesort o la successió de fibonacci. En aquests, no s'aconsegueix informació de timing fins que s'ha arribat a la granularitat més petita, és a dir, el màxim de recursivitat de l'algoritme.

Quan un d'aquests dos escenaris impedeix tenir informació de timing, ens interessa tenir alguna forma de limitar el nombre de tasques com per exemple en fibonacci, on es poden arribar a crear moltes tasques de granularitat fina. La millor manera doncs, és limitar el nombre de tasques comptabilitzant tasques d'un mateix tipus i per fer-ho possible necessitem indicadors que ens ajudin a esbrinar el nivell de recursivitat en el que ens trobem i l'aritat d'un cert tipus de tasca. Aquests indicadors es fan servir per tenir una mitjana del nombre de tasques creades fins al moment i aquesta mitjana, si sobrepassa el nombre màxim de tasques que es calcula amb el nombre de CPUs i el màxim nombre de tasques per CPU (que pot ser modificat si és necessari amb la variable d'entorn `TASKS_PER_CPU`), provoca que la tasca generada sigui final.

La segona part del codi fa referència a quan sí que es té accés a informació de timing del tipus de tasca. En aquest cas, amb la informació del cost de la tasca proporcionada pel programador i la informació de timing normalitzada, és a dir el cost unitari, s'estima el temps d'execució de la tasca i, si no arriba al threshold de temps establert, la tasca es genera automàticament com una tasca final. Aquest threshold pot ser controlat amb la variable d'entorn `THRESHOLD`, tanmateix, el runtime fa servir el temps d'una tasca buida, és a dir, el temps que tarda una tasca a crear-se i destruir-se, per crear un threshold adequat. El temps d'execució d'una tasca per finalitzar la funció de codi d'usuari doncs, hauria de ser almenys 'X' vegades el temps que tarda a gestionar-se una tasca, pel que hauria de sobrepassar el threshold si es vol considerar una tasca amb suficient pes per ser executada sense la clàusula final.

Pel que fa a la condició `!task->hasWeakDepend()` que es pot veure en el codi complet, més endavant s'explicarà el perquè i la problemàtica que va sorgir entre la clàusula final i les dependències weak.

9.4.3 Modificacions a conseqüència del Mòdul Autofinal

Per tal de completar el disseny dels dos mòduls doncs, s'havien d'afegir dues variables extres a la classe `TaskMonitoringInfo`, una pel comptador de tasques que almenys tenen una subtasca i l'altre pel comptador de tasques creades per aquest tipus de tasca. A part, s'afegeixen mètodes per incrementar ambdues variables en el mòdul de monitoratge.

Un cop creades les variables, es va afegir l'ús d'elles i es van incrementar on feia falta. Quan es crea una tasca, a partir d'ara s'han de comprovar 3 situacions:

- La tasca pare que crea aquesta tasca estava marcada com a tasca pare, és a dir, aquesta tasca és la primera creada per la tasca pare? Si no estava marcada, cal marcar-la com a tal ja que acaba de transformar-se en tasca pare, i incrementar la variable (`parentTasks`) del tipus de tasca del pare. Aquesta 'marca' s'ha de dur a terme ja que s'ha de fer una distinció de les tasques que no són pares amb les que sí que ho són per tal de tenir una bona mitjana de l'aritat d'un cert tipus de tasca.

- Incrementar la variable `childrenTasks` del tipus de tasca al qual pertany la tasca pare.

- Si la tasca que s'està creant té la mateixa label que el pare i per tant són del mateix tipus, donar-li a la tasca la profunditat en recursivitat del pare incrementada en 1. Per contrari, si no són del mateix tipus, donar-li a la variable de la profunditat com a valor 0.

En la figura 12 es poden visualitzar millor aquestes tasques o comprovacions a dur a terme al crear una tasca.

```
void nanos_submit_task(void *taskHandle)
{
    /* ... */

    task->setParent(parent);

    if (!parent->isMarkedAsParent()) {
       parent->markAsParent();
       Monitoring::increaseParentTasks(parent);
    }
    Monitoring::increaseChildrenTasks(parent);

    if (parent->getLabel() == task->getLabel()) {
       task->setRecursiveDepth(parent->getRecursiveDepth() + 1);
    }
    else {
        task->setRecursiveDepth(0);
    }

    /* ... */
}
```

Figura 12: *Comprovacions al crear una tasca a causa del mòdul autofinal.*

9.4.4 Implementació del Mòdul Autofinal

El mòdul Autofinal conté 3 variables per controlar la disponibilitat del mòdul i els límits o thresholds del runtime. A part d'això, conté mètodes per controlar aquestes variables i obtenir els valors.

- `AUTOFINAL_ENABLE`: Controla la disponibilitat del mòdul autofinal. Amb aquesta variable es pot activar o desactivar el mòdul.

- `TASKS_PER_CPU`: Estableix un límit de tasques actives per CPU que es farà servir per condicionar si noves tasques es creen com a tasques final. El valor per defecte és 4 tasques per CPU.

- `THRESHOLD`: Estableix un límit de temps d'execució en una tasca que es farà servir per condicionar si noves tasques es creen com a tasques final. El valor per defecte són 100 microsegons i es fa servir sempre i quant s'indiqui pel programador i el runtime no pugui establir un threshold automàticament creant tasques buides.

10 Resultats

10.1 Overhead Afegit pel Mòdul de Mesura

Abans d'entrar en detall en el rendiment aconseguit gràcies a les prediccions, la clàusula cost i el mòdul autofinal, s'havia d'evaluar i comprovar quin era l'overhead afegit pel monitoratge de tasques. Per fer aquest i posteriors anàlisis, s'han usat quatre algoritmes o benchmarks, cada un amb una certa granularitat de tasques. En el plot 13 es visualitzen els resultats obtinguts executant els algoritmes amb i sense el mòdul de mesura activat.

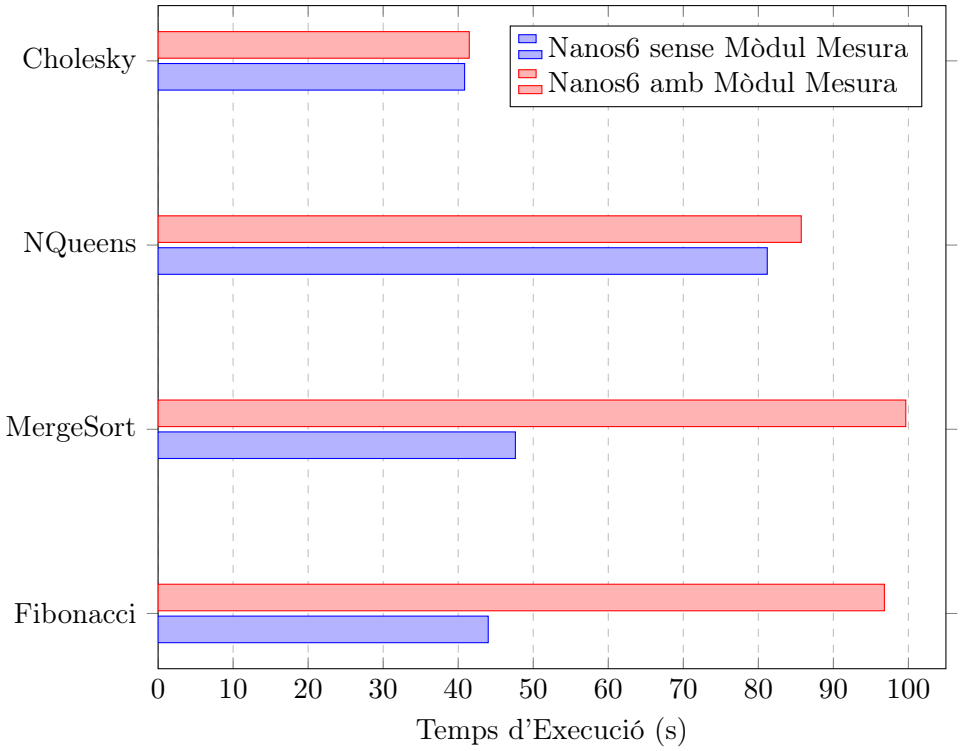

Figura 13: *Overhead afegit mesurat amb diferents benchmarks*

Cal aclarir abans de tot que tots els resultats obtinguts són extrets de mitjanes d'execucions en MareNostrum3 i Mont-Blanc. Els recursos hardware de MareNostrum3 s'han definit en seccions anteriors, tanmateix, cal dir que totes les execucions portades a terme per tal d'extreure resultats i fer anàlisis de rendiments es van portar a terme amb un sistema de cues que assigna 16 CPUs a cada job en el cas de MareNostrum3 i 96 cores i 1 thread per tasca en el cas de Mont-Blanc, per tal que les execucions no tinguin interferències degudes a altres usuaris. A part, cap algoritme s'ha executat amb clàusula final o mòdul autofinal, ja que la finalitat d'aquest primer anàlisi és quantificar el pes del mòdul de mesura en les execucions.

Per tal d'analitzar el rendiment amb una arquitectura diferent, es van replicar tots els resultats i execucions en Mont-Blanc. Des d'Octubre del 2011, l'objectiu del projecte Europeu anomenat Mont-Blanc ha sigut dissenyar un nou tipus d'arquitectura de computadors capaç de crear nous estàndards globals de High Performance Computing basats en solucions energèticament eficients usades en sistemes encastats i dispositius mòbils. Un dels clústers de Mont-Blanc s'anomena *thunder* i és el que es farà servir per analitzar el rendiment.

Thunder disposa dels següents recursos:

- **2x Sockets Cavium ThunderX**
- **48x ARMv8-A cores per socket (96 cores amb memòria compartida) @ 1.8GHz**
- **2x 10GbE (xarxa de dades)**
- **1x 40GbE (no connectada)**
- **1x 128GB SSD**

Pel que fa als algoritmes executats per calcular l'overhead, tot seguit s'introdueixen breument els algoritmes i les mides de l'entrada, que no seran les mateixes pel posterior anàlisi en la secció 10.2.

- **Fibonacci:** La successió de Fibonacci és una successió matemàtica de nombres naturals tal que cada un dels seus termes és igual a la suma dels dos anteriors, cosa que permet fer anàlisi d'overhead en el runtime amb tasques de granularitat molt fina. Pel nostre cas, s'ha executat usant com a índex **35**, és a dir, la successió de fibonacci(35).

- **MergeSort:** MergeSort, algoritme inventat per John von Neumann en 1945, és un algoritme eficient, de propòsit general i basat en comparació per ordenació de vectors.

  Aquest algoritme, per crear el plot 13, es va aplicar a un vector de **10 milions** d'elements sense cap tipus de blocking o threshold en el nivell de recursivitat, no obstant això, permet definir un límit de recursivitat pel que més endavant ens servirà per tenir un algoritme on les tasques poden tenir granularitat variable.

- **NQueens:** Aquest problema consisteix a posar N dames d'escacs en un escaquer (N x N caselles) de tal manera que cap d'elles sigui capaç de capturar-ne qualsevol altra amb els moviments estàndards de la dama dels escacs. S'explica en detall tot seguit en la secció 10.2. La mida de problema era d'un taulell de **15x15**.

- **Cholesky:** La descomposició de Cholesky d'André-Lous Cholesky aprofita que una matriu simètrica definida positiva pot ser descomposta com el producte d'una matriu triangular inferior i la transposada de la matriu triangular inferior. Aquest algorisme permet resoldre diversos sistemes d'equacions lineals. Aquesta aplicació utilitza els algorismes lineals: `potrf`, `trsm`, `syrk` i `gemm` de la llibreria d'Intel d'algorismes matemàtics KML. Per aquest cas, les matrius eren de **14000 x 14000** elements i la mida de bloc de l'algoritme era de **1000 x 1000**. Aquesta mida de block es va escollir provant diferents mides i veient que era la més eficient.

Com es pot veure, en algoritmes on la granularitat de les tasques arriba a ser molt fina com són Fibonacci o MergeSort, el temps afegit pel mòdul de mesura és significatiu atès que el runtime és estressat per tal de monitorar les tasques. Tanmateix, en algoritmes on les tasques tenen una granularitat gruixuda, com NQueens o Cholesky, el monitoratge de tasques és negligible, sobretot en l'últim cas. Així doncs, un cop analitzat l'overhead afegit pel mòdul de mesura i tenint en compte que és negligible en els escenaris de tasques on la granularitat no és fina i que, aquest escenari és justament el que s'intenta reproduir amb la clàusula final i per tant també amb el mòdul autofinal, es pot passar a analitzar el rendiment del runtime extés.

10.2 Rendiment del Runtime Extés

Per tal de mesurar el rendiment aconseguit amb l'extensió del runtime, primer es va començar amb algoritmes on les tasques tenen granularitat fina. El primer d'aquests es Fibonacci. En el plot 14 es poden veure els speedup d'execucions de fibonacci amb índex 30 amb el runtime Nanos6 sense el mòdul autofinal, en relació a l'execució de fibonacci(30) amb el mòdul autofinal activat. Tots aquests speedups són obtinguts d'execucions a MareNostrum3. En el plot 15 es poden veure els mateixos speedups però aquest cop les execucions eren a Mont-Blanc.

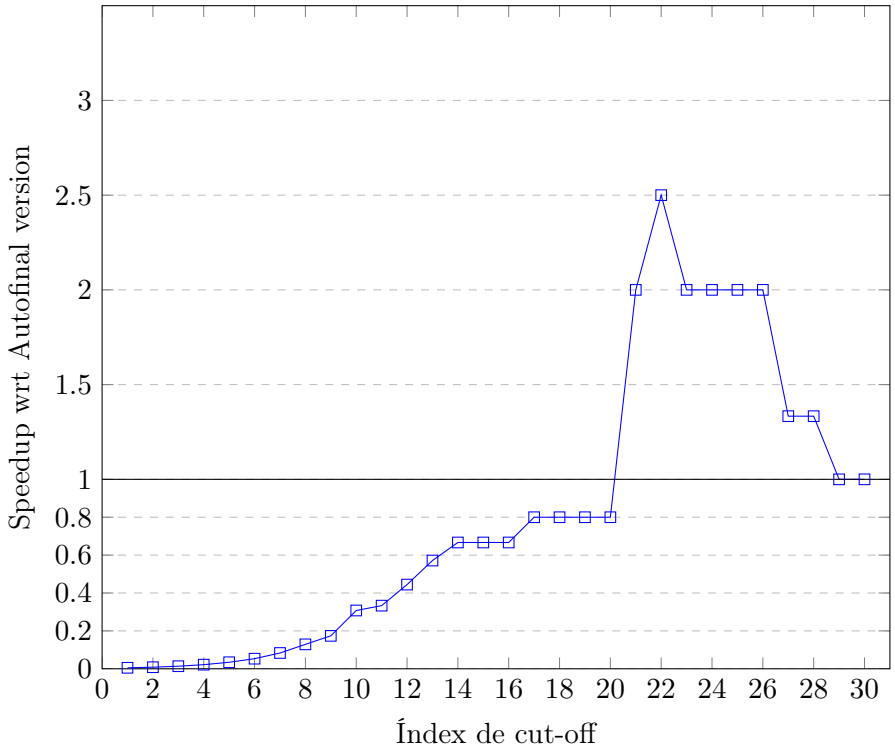

Figura 14: *Fibonacci(30) executat a MN3*

En l'eix de les 'Y' es representa l'speedup (o slowdown en alguns casos) aconseguit en relació a la versió Autofinal (***w**ith **r**espect **t**o Autofinal version*) i en l'eix de les 'X' es representa el valor de tall de la clàusula final que el programador hauria de posar explícitament. Aquest valor de tall indica a quin nivell de recursivitat l'algoritme hauria de començar a generar tasques final, és a dir, si una tasca fibonacci(14) és final, el nivell de recursivitat màxim aconseguit serà 16 (30-14) ja que totes les tasques posteriors no seran generades. Això es pot veure en la figura 16, on es mostra la part de l'algoritme on es veu quines tasques serien final. La versió amb la qual es comparen els resultats per treure l'speedup simplement tenia el mòdul autofinal activat i no tenia la clàusula final en l'algoritme de la figura.

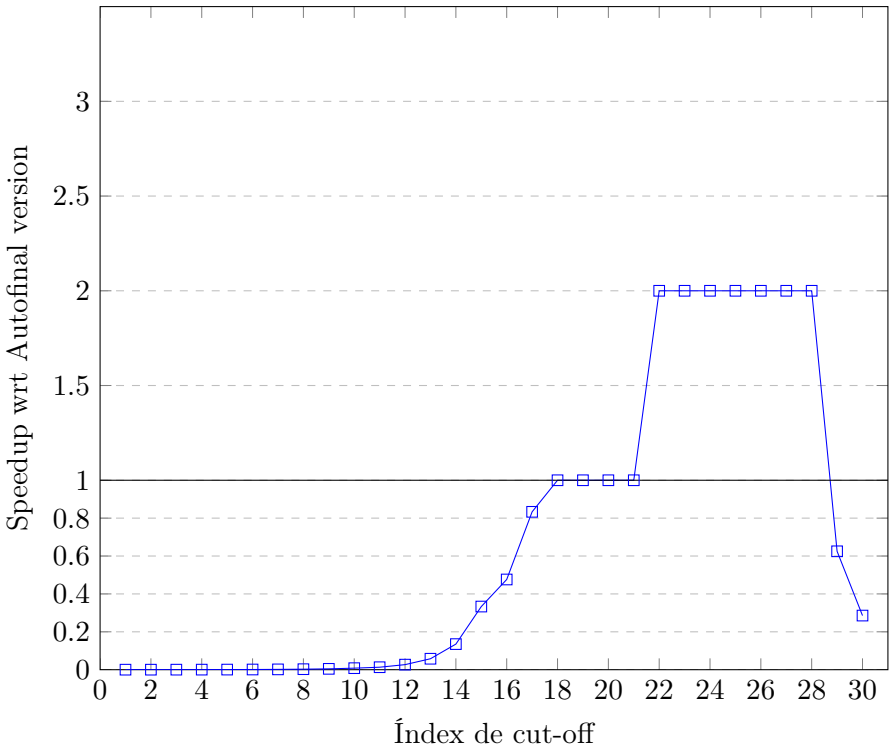

Figura 15: *Fibonacci(30) executat a Mont-Blanc*

```
#define FINAL 14
void fibonacci(int index, int *resultPointer) {
  if (index <= 1) {
    *resultPointer = index;
    return;
  }
  int result1, result2;

  #pragma oss task label(fibonacci) cost(pow(2,index-1)) final(index-1 <= FINAL)
  fibonacci(index-1, &result1);

  #pragma oss task label(fibonacci) cost(pow(2,index-2)) final(index-2 <= FINAL)
  fibonacci(index-2, &result2);

  #pragma oss taskwait
  *resultPointer = result1 + result2;
}
```

Figura 16: *Algoritme de Fibonacci amb clàusula final explícita.*

Per definició d'speedup doncs, totes les mesures per sota de final amb índex 21 en els dos plots són pitjors que l'execució mesurada amb el mòdul autofinal, ja que no hi ha speedup (speedup < 1). Pel que fa a les mesures per sobre d'1, l'speedup aconseguit és bastant pronunciat, ja que els temps d'execució eren de l'ordre de mil·lisegons i el mòdul de mesura afegeix bastant overhead per tasques de granularitat tan petita, com s'ha vist en la secció 10.1. Per tenir una idea de la magnitud en MN3, l'execució amb autofinal tardava 4 mil·lisegons mentre que la millor execució possible amb clàusula final explícita era de 1.6 mil·lisegons.

Així doncs ens interessa veure resultats amb altres algoritmes on la granularitat no sigui tan fina com per exemple mergesort on es poden limitar el nombre d'elements a ordenar. En el plot 17 es poden veure els speedups obtinguts d'execucions de mergesort sobre un vector de 100 milions d'elements a MareNostrum3 i en el plot 18 el mateix pero executant en Mont-Blanc. L'eix de les 'Y' representa l'speedup obtingut respecte a l'execució amb el mòdul autofinal activat i l'eix de les 'X' representa la granularitat de les tasques on es comencen a crear tasques final. Aquesta granularitat és mesurada en elements a ordenar.

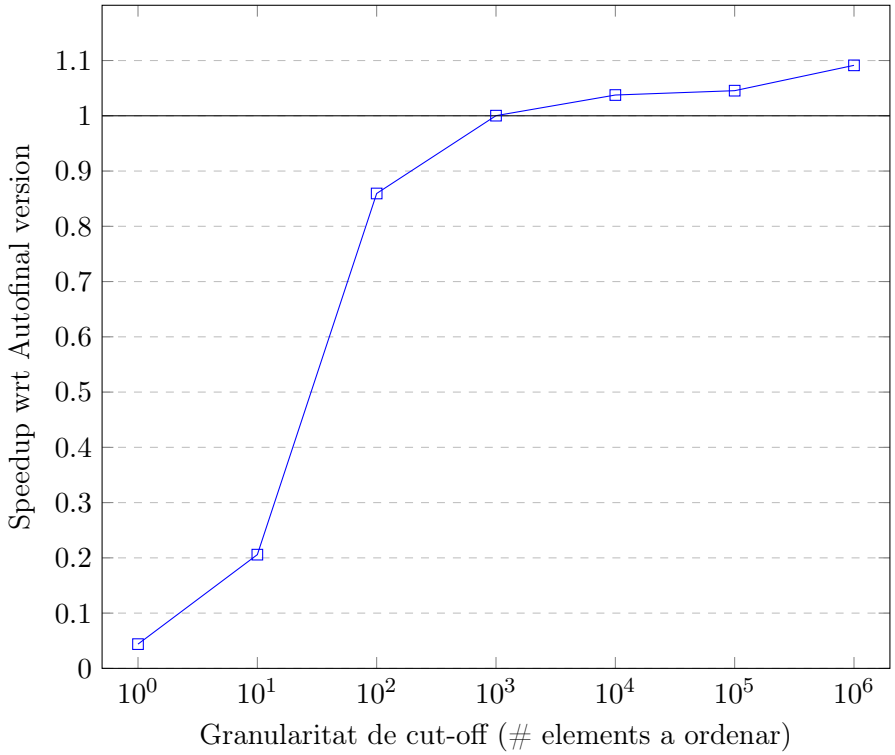

Figura 17: *MergeSort(100.000.000) executat a MN3*

És visible que el rendiment assolit pel runtime extés per aquest algoritme arriba quasi al llindar de la millor execució amb final explícit, ja que l'speedup obtingut amb clàusules final explícites és de 1.1 i 1.3 com a molt en els dos plots. Això és a causa de la granularitat de les tasques que en comparació amb el mòdul de mesura fa que aquest últim sigui quasi negligible pel que fa a temps afegit. Per arribar a tenir una idea de la diferència en granularitat, l'execució en MN3 amb autofinal activat donava un temps d'execució de 42.79 segons mentre que la millor execució amb clàusula final que dóna un speedup de quasi 1.1, donava un temps d'execució de 40.93 segons. Per a tasques amb granularitats més grans del que es mostra en el plot, l'speedup era inferior que aquesta última dada, segurament pel fet de crear masses poques tasques o, el que és el mateix, ser 10^6 la mida més adequada de separació o blocking. Aquesta degradació de rendiment es veu clarament en les execucions portades a terme a Mont-Blanc.

En la figura 19, es mostra el codi usat pels resultats obtinguts. Cal afegir que l'única modificació a aquest algoritme que es va fer amb la versió autofinal és la mateixa que en fibonacci, és a dir, es van eliminar les clàusules explícites i es va activar el mòdul autofinal al compilar el runtime. Pel que fa a la crida merge, és la causant de fer la unió de dues separacions d'un vector ja ordenades individualment, pel que no genera cap subtasca i és bàsica i per tant no es mostrarà la seva implementació.

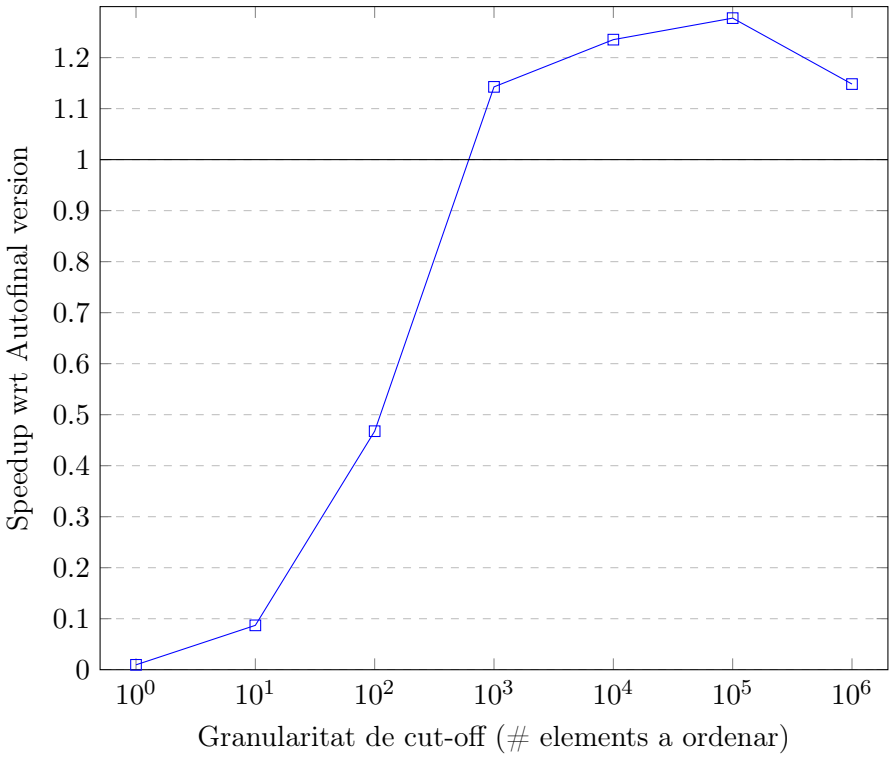

Figura 18: *MergeSort(100.000.000) executat a Mont-Blanc*

```
#define FINAL 100000

void merge_sort(double *a, unsigned long long start, unsigned long long end)
{
  if (start >= end) {
        return;
  }

  unsigned long long mid = (start + end) / 2;
  unsigned long long n1 = (mid - start) + 1;
  unsigned long long n2 = (end - mid);

  #pragma oss task label(merge_sort) cost(n1*log2(n1)) final(n1 <= FINAL)
  merge_sort(a, start, mid);   //Recursive first half

  #pragma oss task label(merge_sort) cost(n2*log2(n2)) final(n2 <= FINAL)
  merge_sort(a, mid + 1, end); //Recursive second half

  #pragma oss taskwait

  /*** MERGE HALVES ***/
  #pragma oss task label(merge) cost((end-start) + 1)
  merge(a, start, end);
}
```

Figura 19: *Algoritme de MergeSort amb clàusula final explícita.*

Per últim es va analitzar el rendiment del runtime usant NQueens, ja que la granularitat de les tasques és fins i tot més gran que en MergeSort.

El problema de les vuit reines (o de les vuit dames) és un problema de raonament lògic que consisteix a posar vuit dames d'escacs en un escaquer (8 x 8 caselles) de tal manera que cap d'elles sigui capaç de capturar-ne qualsevol altra amb els moviments estàndards de la dama dels escacs. Les dames s'han de col·locar de tal manera que no n'hi hagi cap capaç d'amenaçar les altres. Per tant, requereix una solució en què no hi hagi dues dames que comparteixin la mateixa fila, columna o diagonal. Aquest problema és un exemple del problema general de les n reines que consisteix a col·locar n dames en un tauler d'escacs n x n, que només té solucions per a n = 1 o n >= 4.

En els plots 20 i 21, l'eix de les 'Y' representa l'speedup (o slowdown en alguns casos) aconseguit en relació a la versió Autofinal i l'eix de les 'X' representa el valor de tall de la clàusula final que el programador hauria de posar explícitament. Aquest valor de tall indica a quin índex de columna l'algoritme hauria de començar a generar tasques final, és a dir, si comença en l'índex 0, només es generarà una tasca, mentre que si es comença en l'índex 15, es generen totes les tasques com si no hi haguera clàusula final.

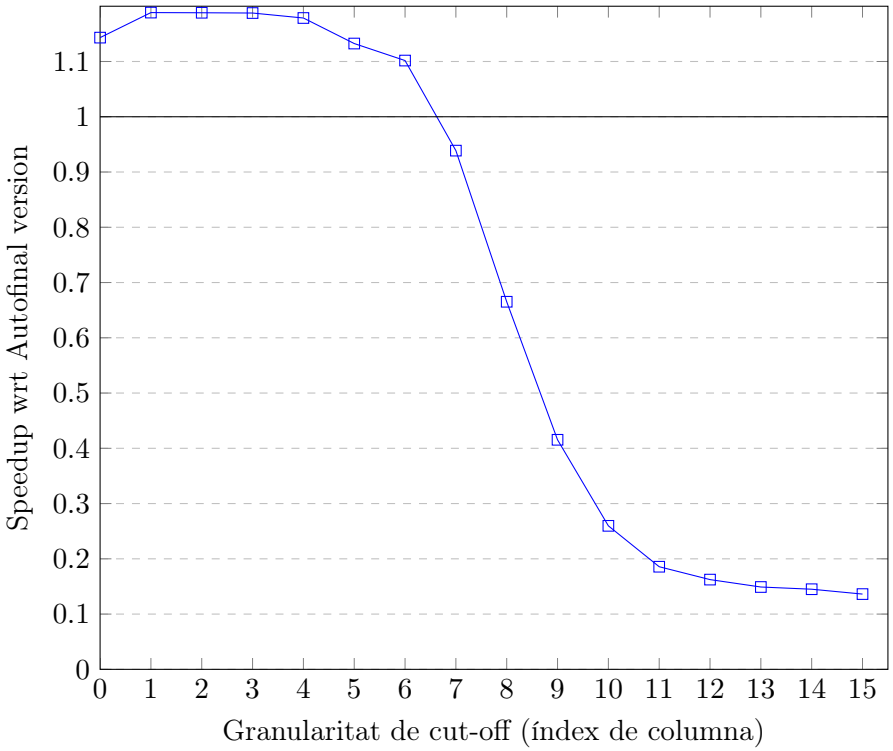

Figura 20: *NQueens(15) executat a MN3*

Com és visible, per a tasques amb granularitat gran el runtime es comporta similar, ja que es pot veure un patró entre els plots de mergesort i nqueens. El mòdul autofinal aconsegueix gairebé el rendiment màxim que s'aconseguiria fent una anàlisi de l'aplicació i variant la clàusula final en diferents execucions. A més, sembla que els resultats aconseguits segueixen un patró de quasi arribar al màxim rendiment, pel que es podria estudiar el valor adequat i afegir més threshold al límit de temps.

Pel plot 21 no s'han considerat cut-offs de més de 8 ja que el temps d'execució incrementava exponencialment, com sol passar amb el problema d'NQueens. La variació en la granularitat es pot veure en les figures 22 i 23, on es mostra la part de l'algoritme on es veu quines tasques serien final. La versió amb la qual es comparen els resultats per treure l'speedup tenia el mòdul autofinal activat i no tenia la clàusula final.

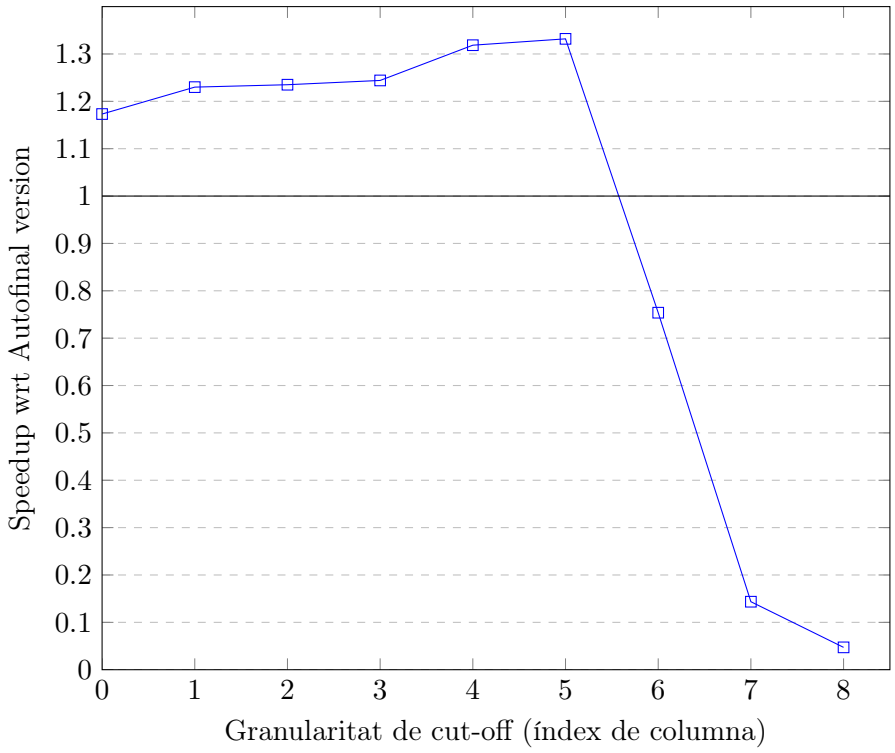

Figura 21: *NQueens(15) executat a Mont-Blanc*

```
#define FINAL 2

void solve(int n, const int col, sol_node_t& sol)
{
    if (col == n) {
        count++;
    }
    else {
        for (int row = 0; row < n; row++)
        {
            if (!check_attack(col, row, &sol)) {
                sol_node_t new_sol;
                new_sol.prev = &sol;
                new_sol.row = row;

                #pragma oss task label(solve) cost(n*(col+1)) final(col <= FINAL)
                solve(n, col + 1, new_sol);
            }
        }
        #pragma oss taskwait
    }
    final_count += count;
    count = 0;
}
```

Figura 22: *Solver de l'algoritme d'NQueens amb clàusula final explícita.*

```c
static inline int check_attack(const int col, const int row, sol_t sol)
{
    for (int j = 0; j < col; j++)
    {
        const int tmp = abs(sol->row - row);
        if (tmp == 0 || tmp == j + 1)
            return 1;

        sol = sol->prev;
    }
    return 0;
}
```

Figura 23: *Mètode de l'algoritme d'NQueens que mira si la posició escollida és amenaçada.*

10.3 Casos d'ús de Nanos6

Com a últim detall en l'anàlisi del runtime, s'ha d'afegir que el principal desenvolupador d'aquest va afegir uns jocs de prova o tests per quan es compila el runtime. En aquests jocs de prova, que són bastants, es proven totes les funcionalitats i casos d'ús del runtime. Aquest tipus de comprovació és molt important fer-la un cop acabat un projecte com aquest, ja que s'ha de comprovar que l'extensió del runtime no hagi modificat el comportament adequat ni introduït errors. En aquests, es comprova l'inicialització del runtime, la creació de tasques i l'aniuament d'aquestes amb l'algorisme de Fibonacci i les dependències, només per nomenar-ne alguns. En la figura 24 es pot veure una captura dels jocs de prova en qüestió un cop es va compilar el runtime extés abans de fer l'anàlisi de rendiment.

```
PASS: lr-nonest.debug.test 124 Evaluating that T4 does not start before T2 finishes
PASS: lr-nonest.debug.test 125 Evaluating that when T4 starts T0 has finished
PASS: lr-nonest.debug.test 126 Evaluating that when T4 starts T2 has finished
PASS: lr-nonest.debug.test 127 Evaluating that when T3 starts T1 has finished
PASS: lr-nonest.debug.test 128 Evaluating that when T3 starts T2 has finished
============================================================================
Testsuite summary for nanos6 1.0
============================================================================
# TOTAL: 3546
# PASS:  3546
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
```

Figura 24: *Captura de pantalla de l'execució dels jocs de prova de Nanos6.*

11 Conclusions

Pel que fa al projecte, s'han assolit tots els objectius que s'havien plantejat al començament. S'ha implementat un mòdul de mesura que, com s'ha pogut veure en els resultats obtinguts, és lleuger pel que fa a temps afegit i aconsegueix fer el monitoratge de totes les tasques. També s'ha extés el model de programació i el runtime per suportar una nova clàusula, la clàusula cost, que ajuda a obtenir el pes relatiu algorítmic de les tasques, i s'ha implementat un mòdul de predicció que fa ús del mòdul de mesura i de la clàusula sempre que hi sigui present, per normalitzar mesures i obtenir una predicció del pes temporal que tindran futures tasques. Per usar aquest mòdul, s'ha dissenyat i implementat un mòdul anomenat autofinal que millora execucions prenent decisions en temps real. Per últim, s'ha analitzat el rendiment del runtime extés amb tots els mòduls i s'ha vist que en general hi ha hagut una millora molt considerable que deriva en estalvi de temps, sigui temps d'execució o temps de feina del personal que fa ús del runtime.

En relació al runtime, s'ha extés tenint en compte tots els requeriments establerts al començament i, com s'ha vist en la secció 10.2, no s'ha modificat el comportament adequat del runtime, és a dir, no s'han introduït errors.

Assolir tots aquests objectius amb 3 dies d'antelació definitivament prova que la planificació estricte i el seguiment continuat han funcionat. El marge temporal previst (5 dies) també s'ha provat suficient ja que només s'han utilitzat 2 dies més del previst.

Personalment, aquest projecte m'ha proporcionat un medi on posar a prova tot el que he anat aprenent en el grau i no només això, també m'ha servit per extendre el meu coneixement i aprendre de personal qualificat en el camp de l'estudi. He extés el meu coneixement en control de versions, gestió de memòria (ús de valgrind, correcció de memory leaks...), paral·lelisme, APIs, C, C++ i he après sobre el model de programació OmpSs i el runtime Nanos6.

12 Treball Futur

En aquesta secció s'introdueixen idees o solucions descartades per manca de recursos temporals però que, tot i així, es continuaran en un futur.

12.1 Separació de tasques per *Buckets* en Cost

Com s'ha comentat en seccions anteriors, el mòdul de monitoratge, per tasques amb costos semblants té prediccions encertades. No obstant això, per tasques amb costos suficientment diferenciats, és possible que el cost normalitzat (és a dir el cost unitari d'un tipus de tasca) sigui diferent. Aquesta diferència pot venir donada per diferents factors però el més obvi és que vingui donada pels nivells de memòria. En la figura 25 es visualitza un exemple del que podria passar per algoritmes amb tasques amb costos diferents.

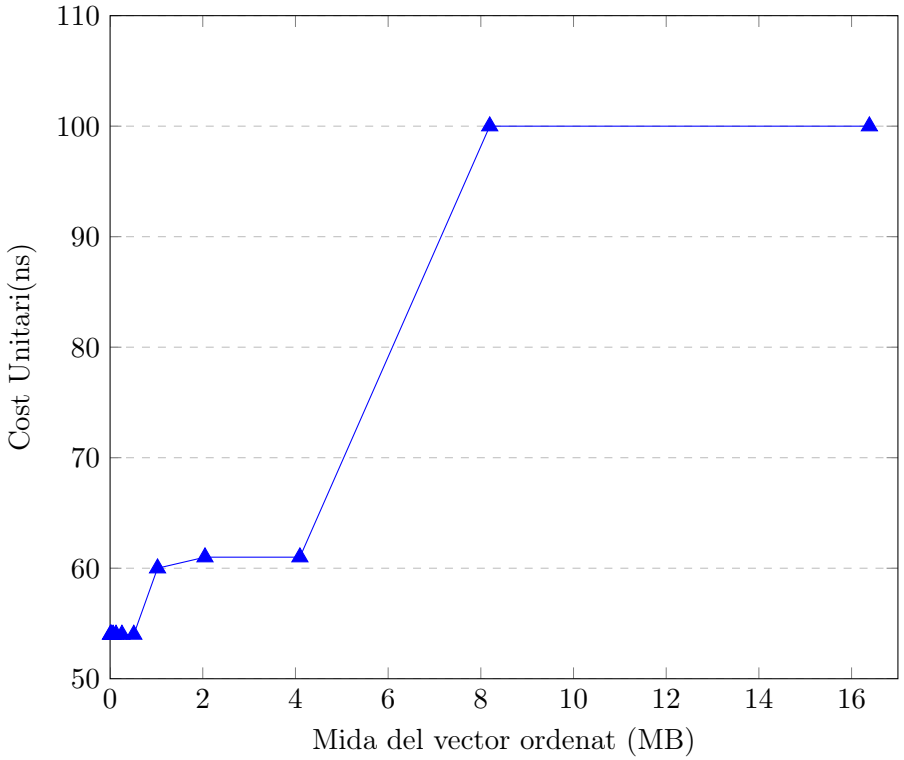

Figura 25: *Gràfic d'execucions de mergesort amb el cost unitari associat*

La figura mostra la mitjana del cost unitari de bastants execucions de mergesort amb diferents mides de vector per ordenar. Com es pot veure, al voltant de les mides on hi ha un canvi de nivell de memòria com Cache2-Cache3 o Cache3-Memòria, hi ha una diferència important en el cost unitari de l'algoritme. Això és visible quan passem d'executar un mergesort de 64KB-512KB, és a dir uns 65000 elements, a executar un mergesort de 1MB (130000 elements). Aquest petit canvi de 54 nanosegons a 61 nanosegons no és gaire significatiu, probablement perquè el canvi de nivell és de Cache2 a Cache3. Tanmateix, si ens fixem en el següent canvi que fa referència al canvi entre Cache3 i Memòria, el cost unitari comença a diferenciar-se bastant. Aquest canvi es pot veure quan passem dels 8MB.

Així doncs, queda bastant clar que la problemàtica és possible que existeixi però com s'ha explicat anteriorment, per l'objectiu d'aquest projecte i considerant que la majoria de benchmarks no varien les mides dels inputs d'algoritmes com mergesort ja que no té sentit, s'ha decidit deixar aquesta separació d'estructures de timing segons rangs de costos per un futur. La solució però, és ben clara, ja que l'única modificació que s'hauria de fer és comparar mitjanes de costos i si varien suficient (desviació estàndard suficientment alta), dividir les estructures en dos buckets diferents, és a dir, dos rangs de costos, i així successivament fins a trobar l'equilibri de costos entre rangs de valors.

12.2 Scheduler

Ja s'ha mencionat que l'objectiu principal per fer servir el mòdul de monitoratge era fer un scheduler que tingués la seva pròpia política d'scheduling usant el cost d'una tasca com a unitat de treball, no obstant això, es deixarà com a treball futur ja que es va considerar que el mòdul autofinal era més interessant i implementar un scheduler no és gens senzill i requereix bastant temps.

12.3 Problemàtica amb Weak Dependencies

En la secció 9.4.2 es podia veure la condició `!task->hasWeakDepend()` abans de decidir si una tasca ha de ser final o no. Aquesta condició va sorgir d'un problema trobar entre aquest projecte i les weak dependencies.

Si una tasca té weak dependencies, això vol dir que alguna de les subtasques que seran creades per la tasca tindran aquestes dependències. El fet que una tasca sigui final però, fa que aquesta no pugui crear altres tasques i que ella mateixa executi tot el codi que pertany a subtasques. Això fa que totes les dependències de tipus weak passin a ser *strong* dependències pel simple fet que passen a ser de la tasca pare. Aquesta conversió de dependències no és gens trivial i s'ha de modificar l'arbre de dependències, pel que per aquest projecte es va decidir no tenir-ho en compte i simplement no tenir en compte tasques amb dependències weak com a potencials tasques final. Així doncs, la solució a aquest problema al qual hem fet un *bypass* es deixa com a treball futur.

13 Revisió del projecte

Amb motiu del mòdul de GEP, en aquesta secció es presenta la revisió de la memòria del projecte final de grau un cop finalitzada la fita inicial.

Marc de legalitat

Com bé s'ha especificat en aquest document, aquest projecte és una modificació d'un projecte *open source* existent. Com a tal, qualsevol persona pot col·laborar en les *release* oficials del projecte Nanos6.

Canvis en la planificació

Pel que fa a la planificació, l'etapa de canvis en les polítiques *d'scheduling* s'ha modificat bastant. Es va detectar que el fet de crear un scheduler des de zero alimentant-lo només de la informació donada per la clàusula cost era una tasca gens trivial i de molta durada, ja que al començar a detallar la implementació van sorgir molts problemes i codi per implementar. Aquesta implementació era la que es volia fer servir des del principi, no obstant això, hi havia més treball futur i, és aquest mateix treball, el que s'ha intercanviat per la tasca. La temàtica de la tasca es detalla a continuació.

Canvis en les polítiques *d'scheduling*: Automatic Final

De l'etapa com a tal no es modifica en absolut ni la planificació temporal ni la planificació de costos i recursos, simplement es canvia el mètode de modificació en les polítiques *d'scheduling*. En comptes de crear un scheduler, s'usarà la informació de la clàusula cost i d'altra informació del sistema per tal de canviar la política *d'scheduling* existent en cas que sigui necessari.

Aquest canvi s'anomena *Automatic Final* i consisteix a detectar en quin moment la granularitat d'una tasca que s'està creant és suficientment insignificant per tal d'executar-la com a tasca "final". La diferència entre una tasca final i una no final és que totes les tasques filles (A', A"...) d'una tasca final (A) no es crearan com tasques, sinó que s'executaran de manera seqüencial pel mateix thread. En altres paraules, el thread encarregat d'executar la tasca A executarà el codi dins de les tasques A', A"... de forma seqüencial i sense encapsular aquesta computació en sub-tasques per tal d'evitar els *overheads* de creació i planificació, que haguessin acabat sent majors que el temps d'execució de la tasca.

Donat que els canvis i la temàtica de l'etapa modificada són idèntics a l'anterior, no es canvia ni el nom, ni el temps dedicat ni els recursos, ja que analitzant totes les alternatives aquesta ha sigut la més raonable a la que s'ha arribat de forma consensuada.

Canvis en la metodologia

En relació a la metodologia, no es va canviar cap mètode de treball, tanmateix, s'ha adoptat una nova sessió de *training*. Donat que el desenvolupador i d'altres treballadors que es trobaven en projectes relacionats tenien alguns dubtes sobre certs aspectes o parts del runtime, l'encarregat d'aquest va decidir crear una sessió d'entrenament on s'expliquen per a tothom com funcionen aquestes parts que no quedaven clares.

Les sessions d'entrenament no són obligatòries, pel que el desenvolupador pot triar a quines vol assistir i a quines no, segons les preferències que tingui. Un exemple de sessió a la qual el desenvolupador va assistir era el funcionament de les polítiques *d'scheduling* de Nanos6.

Aquestes sessions no incrementen el temps de realització del projecte, ja que són meres explicacions que, en qualsevol cas, s'haurien d'haver fet en reunions privades ja comptabilitzades en el diagrama de Gantt.

14 Referències

[1] OpenMP.org (2015, Novembre), *OpenMP Application Programming Interface, Version 4.5*, Recuperat de `http://www.openmp.org/mp-documents/openmp-4.5.pdf`.

[2] Labarta ,J., Barcelona Supercomputing Center (BSC), *StarSs: a Programming Model for the Multicore Era*, Recuperat de `http://www.prace-ri.eu/IMG/pdf/08_starss_jl.pdf`.

[3] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), *Descripció del compilador Mercurium*, Recuperat de `https://pm.bsc.es/mcxx`.

[4] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), *Descripció del runtime Nanos++*, Recuperat de `https://pm.bsc.es/nanox`.

[5] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), *Descripció del model de programació OmpSs*, Recuperat de `https://pm.bsc.es/ompss`.

[6] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), *Paraver, Performance Analysis Tools: Details and Intelligence*, Recuperat de `http://www.bsc.es/computer-sciences/performance-tools/paraver`.

[7] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), *Dimemas: predict parallel performance using a single cpu machine*, Recuperat de `https://www.bsc.es/computer-sciences/performance-tools/dimemas`.

[8] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), *Extrae, Performance Tool*, Recuperat de `https://www.bsc.es/computer-sciences/extrae`.

[9] Chamberlain, B. (2013, Maig 14), *Chapel: Productive Parallel Programming*, Recuperat de `http://www.cray.com/blog/chapel-productive-parallel-programming`.

[10] Radigan, Dan *How the kanban methodology applies to software development*, Recuperat de `https://www.atlassian.com/agile/kanban`.

[11] *DAD: A Process Decision Framework*, Recuperat de `http://www.disciplinedagiledelivery.com/introduction-to-dad/`.

[12] (2013, Agost 27), *Extreme Programming for Extreme Programmers*, Recuperat de `https://airbrake.io/blog/internet/extreme-programming-for-extreme-programmers`.

[13] *Qúe es SCRUM*, Recuperat de `https://proyectosagiles.org/que-es-scrum/`.

[14] *GIT, system version control*, Recuperat de `https://git-scm.com/about`

[15] *Gitlab, Tools for modern developers*, Recuperat de `https://about.gitlab.com/`

[16] *Web oficial de Microsoft OneNote*, Recuperat de https://www.onenote.com/

[17] *Web oficial de l'aplicació sharelatex*, Recuperat de https://www.sharelatex.com/about

[18] *Web oficial Dropbox*, Recuperat de https://www.dropbox.com

[19] *Web oficial de l'aplicació Gantter*, Recuperat de https://gantter.com/

[20] *Web del producte mencionat, Laptop Dell*, Recuperat de http://www.dell.com/us/business/p/latitude-e7450-ultrabook/pd

[21] *Web del producte mencionat, Monitor Dell*, Recuperat de http://www1.euro.dell.com/es/es/domestica/Dell-Perifricos/dell-p2715q-monitor/pd.aspx?refid=dell-p2715q-monitor&cs=esdhs1&s=dhs

[22] *Web oficial Moto G4*, Recuperat de https://www.motorola.es/products/moto-g

[23] Estudi de remuneració Michael Page, *Estudi de remuneració general Michael Page a Espanya l'any 2016* [Consulta: 9 d'Octubre de 2016], Recuperat de http://www.michaelpage.es/sites/michaelpage.es/files/ingenieros2016.pdf.

[24] Microsoft Windows 10 Home, *Web oficial de Microsoft* [Consulta: 8 d'Octubre de 2016], Recuperat de https://www.microsoftstore.com/store/mseea/es_ES/pdp/Windows-10-Home/productID.320437800?s_kwcid=AL!4249!3!138661986204!e!!g!!microsoft%20windows%2010&WT.mc_id=pointitsem+Google+Adwords+Windows+10+-+ES&ef_id=V9bUiAAABSrisj9H:20161008214957:s

[25] Microsoft Office 365 Universitarios, *Web oficial de Microsoft* [Consulta: 8 d'Octubre de 20166], Recuperat de https://www.microsoftstore.com/store/mseea/es_ES/pdp/Office-365-Universitarios/productID.263156100?ICID=Office_365_ModF_365University.

[26] Motorola Moto G4 Plus, *Botiga online FNAC* [Consulta: 8 d'Octubre de 2016], Recuperat de http://www.fnac.es/mp5079319/Motorola-Moto-G4-Plus-16Go-Dual-SIM-Negro?origin=SEA_GOOG_PLA_MK_TEL&gclid=CjwKEAjwsuK_BRDD9ISR1bawwUwSJACbOiixLT8fnwBOWF_OXIOthmRLNmmCXVrUd_r5mf_EC2ko3BoCTqTw_wcB&gclsrc=aw.ds.

[27] Bitllet integrat TJove (3 mesos) de 6 zones, *Pàgina web de TMB* [Consulta: 9 d'Octubre de 2016], Recuperat de https://www.tmb.cat/es/barcelona/tarifas-metro-bus/abonos/t-jove#hdr_preus_zona.

[28] Definició dels *Internet Relay Chats*, *Article Wikipedia sobre els IRC*, Recuperat de https://es.wikipedia.org/wiki/Internet_Relay_Chat.

[29] Fernando J. Pérez i Pere Ríos, (2014, Novembre 10), *Consulta sobre la independència de Catalunya*, Recuperat de http://politica.elpais.com/politica/2014/11/09/actualidad/1415542400_466311.html.

Apèndixs

	Name	Start	Finish	Predecessors
1	**⊟Gestió del projecte**	**19/09/2016**	**24/10/2016**	
2	Abast del projecte i contextualització	19/09/2016	27/09/2016	
3	Planificació temporal	27/09/2016	03/10/2016	2
4	Gestió econòmica i sostenibilitat	03/10/2016	10/10/2016	3
5	Presentació preliminar	10/10/2016	17/10/2016	4
6	Mòdul d'enginyeria de computadors	17/10/2016	24/10/2016	5
7	Presentació oral i document final	17/10/2016	24/10/2016	5
8	Fita: Document Final GEP	24/10/2016	24/10/2016	
9	Estudi del model de programació i el runtime	25/10/2016	27/10/2016	1
10	Anàlisi general	28/10/2016	01/11/2016	9
11	Fita: Presentació Oral GEP	11/11/2016	11/11/2016	
12	Creació de l'entorn	28/10/2016	03/11/2016	9
13	**⊟Disseny i implementació**	**04/11/2016**	**11/01/2017**	**10,12**
14	**⊟Mòdul de Mesura**	**04/11/2016**	**16/11/2016**	
15	Disseny	04/11/2016	08/11/2016	10,12
16	Implementació i Tests	09/11/2016	16/11/2016	15
17	**⊟Clàusula cost**	**04/11/2016**	**08/11/2016**	
18	Disseny	04/11/2016	07/11/2016	10,12
19	Implementació i Tests	07/11/2016	08/11/2016	
20	**⊟Mòdul de predicció**	**17/11/2016**	**02/01/2017**	
21	Disseny	17/11/2016	21/11/2016	14,17
22	Implementació i Tests	22/11/2016	02/01/2017	21
23	**⊟Mòdul Autofinal**	**03/01/2017**	**11/01/2017**	
24	Disseny	03/01/2017	05/01/2017	22
25	Implementació i Tests	06/01/2017	11/01/2017	24
26	Avaluació del Rendiment	12/01/2017	13/01/2017	25
27	Memòria Final	14/01/2017	16/01/2017	26
28	Fita: Entrega Memòria Final	16/01/2017	16/01/2017	
29	Fita: Defensa Oral TFG	23/01/2017	27/01/2017	

Figura 26: *Taula de Gantt amb les dates d'inici i finalització, la durada, els nivell de riscs i els recursos de les tasques del projecte.*

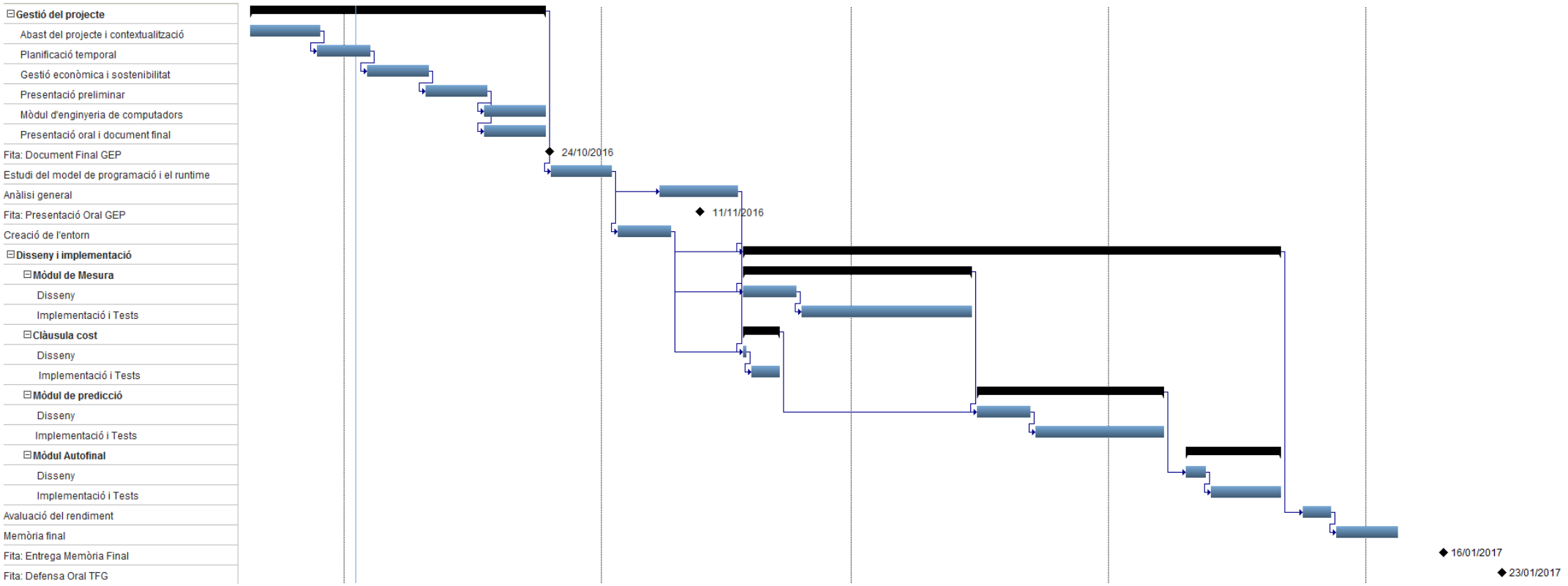

Figura 27: *Diagrama de Gantt amb les tasques del projecte, les dependències entre elles i fites de temps relacionades amb el projecte.*

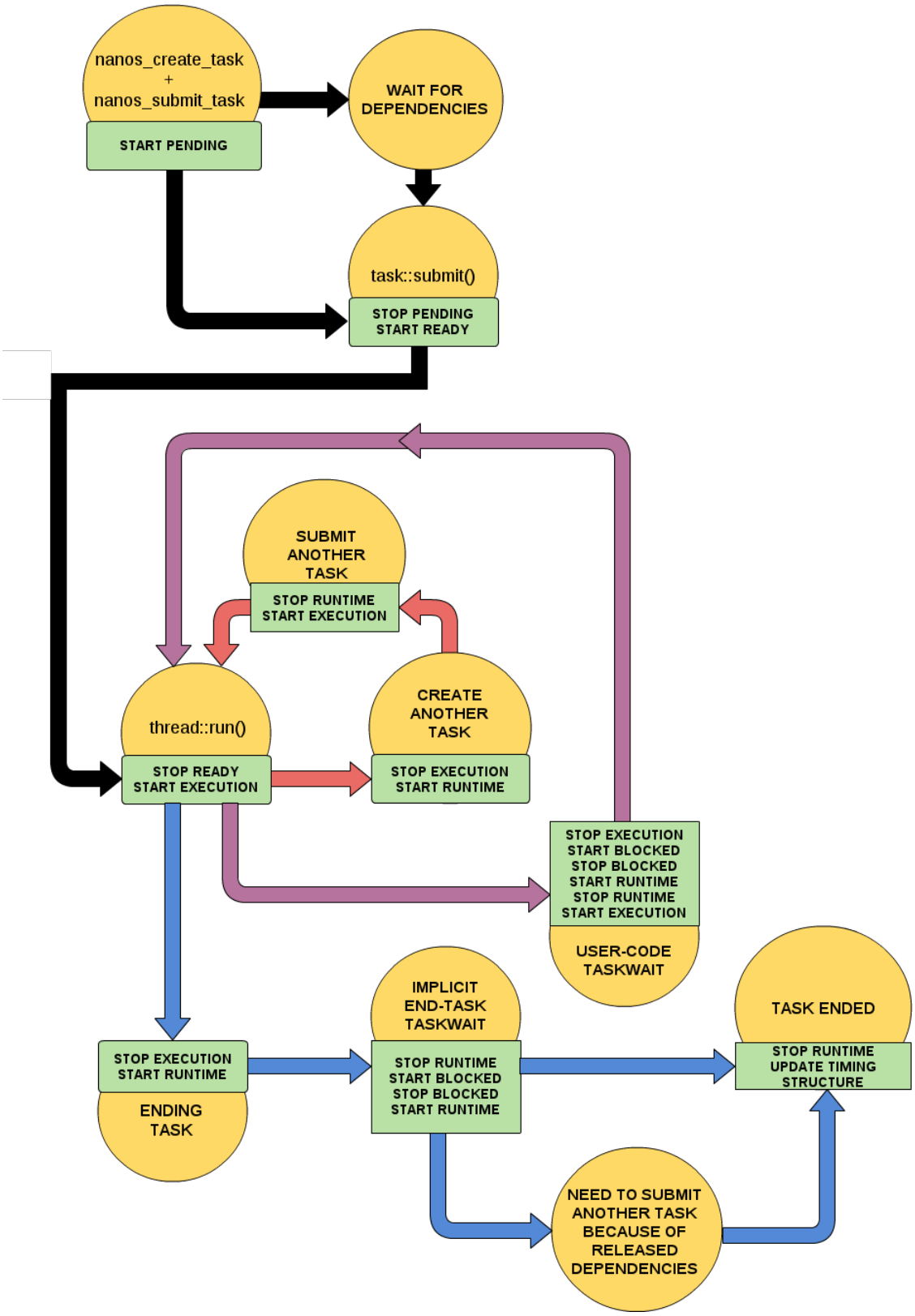

Figura 28: *Diagrama del pas d'estats en tasques per tal d'obtenir mètriques de timing.*